Faculta de Ingeniería

Escuela de Computación (EIC)

Desarrollo de Aplic. Web con Soft. Intérprete en el Servidor DSS404 G05T

“Investigación aplicada 2”

Docente: René Mauricio Tejada Tobar

Integrantes:

Caleb Alejandro Peñate Deras	PD230166
Diego Alberto Canizalez Galán	CG232300
Camila Elizabeth Castillo Joya	CJ220498
Daniel Adonay García Aguilar	GA232128

Introducción

En el desarrollo web contemporáneo, las aplicaciones deben ser capaces de responder en tiempo real, manejar múltiples eventos simultáneamente y escalar eficientemente para satisfacer la demanda de los usuarios. El paradigma de programación orientado a eventos (*event-driven*) ha emergido como una solución clave para alcanzar estos objetivos, al permitir una ejecución asincrónica y no bloqueante. Si bien este modelo ha sido ampliamente adoptado en entornos como Node.js, su integración en PHP ha sido más reciente, impulsada por el surgimiento de herramientas como Swoole y ReactPHP, que permiten transformar el comportamiento tradicional de PHP hacia una arquitectura más reactiva y persistente.

Históricamente, PHP ha operado bajo un modelo síncrono y basado en procesos por cada solicitud, lo que limita su capacidad para manejar cargas concurrentes de forma eficiente. Sin embargo, mediante el uso del paradigma event-driven y tecnologías de infraestructura moderna —como la horizontalización de servicios, el uso de contenedores (Docker), balanceadores de carga, y orquestadores como Kubernetes— es posible escalar aplicaciones PHP de manera similar a otros lenguajes modernos.

Esta investigación tiene como objetivo analizar la implementación del paradigma event-driven en PHP, sus beneficios, herramientas disponibles, y cómo escalar estas aplicaciones en entornos distribuidos. Además, se compararán casos reales con modelos basados en hilos, se discutirán los desafíos técnicos, las mejores prácticas y se propondrán futuras líneas de investigación para seguir ampliando el potencial de PHP en el desarrollo de aplicaciones web modernas.

Definición del Paradigma Event-Driven:

La programación basada en eventos (Event-Driven Programming) es un paradigma de desarrollo en el que el flujo de ejecución de un programa está determinado por eventos, como acciones del usuario (clics, teclas), mensajes de otros procesos o señales del sistema.

En lugar de seguir un flujo secuencial predefinido, el programa reacciona a eventos de manera asíncrona, ejecutando funciones de callback o manejadores (event handlers) cuando ocurren dichos eventos.

Importancia en el Desarrollo de Aplicaciones Web

En el desarrollo web moderno, este paradigma es esencial porque:

- Permite manejar interacciones del usuario en tiempo real (ej: websockets, formularios).
- Facilita la escalabilidad en servidores, ya que evita bloquear el hilo principal mientras se esperan operaciones de E/S (como consultas a bases de datos o APIs).
- Es la base de tecnologías como Node.js, WebSockets y frameworks como React (donde el UI responde a eventos).

Principios Fundamentales del Paradigma Event-Driven

-Asincronía y No Bloqueo (Non-Blocking I/O)

Las operaciones no detienen la ejecución del programa. En su lugar, se registran callbacks que se ejecutan cuando la operación termina.

Ejemplo: Un servidor web puede atender múltiples solicitudes mientras espera respuestas de una base de datos.

-Event Loop (Bucle de Eventos)

Es el núcleo de entornos como Node.js. Monitorea eventos en cola y ejecuta sus manejadores cuando los recursos están disponibles.

Permite alta concurrencia sin necesidad de múltiples hilos.

-Emisión y Escucha de Eventos (Event Emitters & Listeners)

Los objetos emiten eventos y otros se suscriben a ellos.

Ejemplo: En un servidor HTTP, se escucha el evento 'request' para manejar peticiones entrantes.

-Escalabilidad y Eficiencia

Al evitar bloqueos, se optimiza el uso de recursos, ideal para aplicaciones con muchas conexiones simultáneas.

Estado Actual de PHP

PHP sigue un modelo síncrono y sin estado (stateless) en su ejecución tradicional, basado en el esquema "request-response":

Proceso por Solicitud (Síncrono):

Cada solicitud HTTP inicia un nuevo proceso (o hilo, dependiendo del servidor web, como Apache con mod_php o PHP-FPM).

El intérprete de PHP ejecuta el script desde cero en cada petición, sin compartir memoria entre solicitudes

Ciclo de Vida de una Solicitud:

- **Inicio:** El servidor web (Apache/Nginx) recibe la petición y la pasa al intérprete PHP.
- **Ejecución:** PHP procesa el script, realiza consultas a BD, genera HTML, etc.
- **Fin:** Envía la respuesta y **termina el proceso**, liberando recursos.

Gestión de Procesos (PHP-FPM):

PHP-FPM (FastCGI Process Manager) mejora el rendimiento administrando un pool de procesos PHP reutilizables, evitando iniciar/cerrar procesos en cada solicitud.

Sin embargo, cada solicitud sigue atendiéndose de manera síncrona dentro de su propio proceso.

Alto Consumo de Recursos por Solicitud:

Cada solicitud requiere memoria y CPU independiente. En entornos con miles de solicitudes simultáneas, esto genera:

- Saturación rápida de la RAM.
- Overhead por creación/terminación de procesos.

Bloqueo de Operaciones de E/S:

Si un script espera una respuesta de una API externa o una consulta lenta a la BD, el proceso PHP queda bloqueado hasta que termine.

Escalabilidad Vertical:

La solución tradicional es aumentar servidores (escalar verticalmente), lo que incrementa costos.

No aprovecha eficientemente el hardware multicore.

Falta de Concurrencia Real:

PHP no tiene un **event loop nativo** lo que limita su capacidad para manejar operaciones asíncronas sin extensiones externas.

Problemas con Conexiones Persistentes:

PHP no está diseñado para mantener conexiones largas (como chats en tiempo real), ya que su modelo "muere" después de cada respuesta.

Alternativas para Mejorar PHP en Cargas Altas

Usar PHP con Swoole:

Swoole es una extensión que añade event loop y corrutinas a PHP, permitiendo manejar miles de conexiones concurrentes (similar a Node.js).

Implementar ReactPHP:

Biblioteca para programación asíncrona en PHP (basada en promesas).

Útil para APIs y microservicios.

Arquitecturas Híbridas:

Delegar tareas pesadas a colas (con RabbitMQ o Redis) y workers asíncronos.

Frameworks y Librerías Event-Driven en PHP

PHP tradicionalmente ha seguido un modelo síncrono y bloqueante, pero gracias a herramientas como Swoole y ReactPHP, ahora puede adoptar un paradigma event-driven, permitiendo alta concurrencia, operaciones asíncronas y aplicaciones en tiempo real

Swoole: PHP Asíncrono y de Alto Rendimiento

-Características Principales

Motor Event-Driven y Corrutinas

Proporciona un event loop similar al de Node.js, permitiendo manejar miles de conexiones en un solo proceso.

Soporta corrutinas (como las de Go), evitando el "callback hell" y mejorando la legibilidad del código.

Servidor HTTP/WebSocket Integrado

Permite crear servidores HTTP y WebSocket sin depender de Nginx/Apache.

Ideal para aplicaciones en tiempo real (chats, juegos, notificaciones push).

Programación Asíncrona para E/S

Operaciones de red, bases de datos y sistema de archivos no bloqueantes.

Alto Rendimiento

Benchmarkings muestran que Swoole puede manejar hasta 10 veces más solicitudes por segundo que PHP-FPM tradicional.

ReactPHP: Event-Driven para PHP Puro

-Características Principales

Event Loop Basado en Libevent

- Implementa un bucle de eventos en PHP puro (sin extensiones).
- Soporta promesas y flujos reactivos para manejar operaciones asíncronas.

Componentes para E/S No Bloqueante

- HTTP, WebSockets, sockets TCP/UDP, sistema de archivos, procesos hijos.

Interoperabilidad con Librerías PHP Existentes

- Puede combinarse con Guzzle (HTTP asíncrono), Doctrine DBAL, etc.

Otras Herramientas Relacionadas

Amp

Similar a ReactPHP, pero con un enfoque más moderno (usa generadores y corrutinas).

RoadRunner

Un servidor app PHP high-performance que usa Goroutines (Go) para gestionar workers PHP.

Ideal para Laravel, Symfony sin modificar código.

FrankenPHP (PHP en Go)

Un servidor web integrado que combina PHP con el runtime de Go para mejor concurrencia.

Comparación con Modelos Basados en Hilos:

Los modelos de concurrencia en aplicaciones web modernas se dividen principalmente en dos paradigmas: event-driven (basado en eventos) y thread-based (basado en hilos). Cada enfoque presenta características distintivas que los hacen adecuados para diferentes escenarios de desarrollo.

Fundamentos Conceptuales

El modelo event-driven opera mediante un bucle de eventos (event loop) que gestiona operaciones de forma no bloqueante en un único hilo de ejecución. Este paradigma es característico de entornos como Node.js, PHP con Swoole/ReactPHP y Python con asyncio.

Por otro lado, el modelo thread-based utiliza múltiples hilos o procesos para manejar solicitudes concurrentes, siendo implementado comúnmente en frameworks como Flask/Django (con Gunicorn), Spring Boot y Ruby on Rails.

Análisis de Rendimiento

En términos de latencia, los sistemas event-driven muestran ventajas significativas (10-50ms) gracias a su naturaleza no bloqueante, mientras que los sistemas thread-based presentan mayor latencia (50-200ms) debido al overhead de planificación de hilos.

El throughput en aplicaciones I/O bound favorece claramente al modelo event-driven, capaz de manejar entre 10,000 y 50,000 solicitudes por segundo, frente a las 1,000-5,000 solicitudes de los sistemas thread-based.

La gestión de memoria revela diferencias sustanciales: mientras una aplicación event-driven puede manejar 10,000 conexiones con aproximadamente 300MB de RAM, un sistema thread-based requeriría alrededor de 8GB para el mismo número de conexiones (asumiendo 1MB por hilo).

Escalabilidad

Los sistemas event-driven destacan en escalabilidad vertical, pudiendo manejar más de un millón de conexiones por instancia cuando están optimizados..

En contraste, los sistemas thread-based suelen alcanzar límites prácticos entre 10,000 y 50,000 conexiones por servidor, como mostraba Instagram antes de migrar a soluciones async.

Complejidad de Desarrollo

El desarrollo en entornos event-driven presenta mayores desafíos:

- Los flujos asíncronos generan estructuras de código más complejas
- El debugging requiere herramientas especializadas debido a stack traces fragmentados
- La propagación de errores se vuelve menos intuitiva

Los sistemas thread-based, por su parte, ofrecen:

- Flujos de ejecución lineales y más fáciles de seguir
- Mejor compatibilidad con debuggers tradicionales
- Menor curva de aprendizaje para desarrolladores junior

6. Casos de Uso Recomendados

Para aplicaciones event-driven:

- APIs de alta demanda (10,000+ req/seg)
- Sistemas en tiempo real (chat, juegos multiplayer)
- Microservicios con predominio de operaciones I/O

Para aplicaciones thread-based:

- Procesamiento intensivo de CPU (ML, ETL)
- Aplicaciones empresariales complejas
- Sistemas legacy que requieren integración sencilla

Ventajas y Desventajas de Modelos Event-Driven y Thread-Based

Modelo Event-Driven

Ventajas:

- Alto rendimiento en E/S (I/O-bound)
- Eficiencia en recursos
- Escalabilidad horizontal más sencilla
- Buen soporte para operaciones asíncronas

Desventajas:

- Complejidad en el código
- Problemas con CPU
- Flujos asíncronos complican el rastreo de errores
- Dependencia de extensiones (en PHP)

Modelo Thread-Based

Ventajas:

- Simplicidad de desarrollo
- Mejor manejo de CPU
- Debugging más fácil
- Compatibilidad con librerías tradicionales

Desventajas:

- problema en altas cargas en memoria y CPU
- Limitación en concurrencia
- Problemas con estados compartidos
- Ineficiente para conexiones persistentes

Desafíos y Consideraciones

Desafíos al implementar Event-Driven en PHP:

- **Compatibilidad con código existente:**
  PHP ha sido históricamente un lenguaje diseñado para aplicaciones web síncronas, donde cada petición inicia y finaliza un nuevo proceso. Al cambiar a un modelo event-driven, donde el servidor permanece activo y escucha eventos de forma continua, muchas estructuras tradicionales dejan de ser compatibles. Esto obliga a refactorizar gran parte del código para adaptarse al nuevo paradigma, especialmente en aplicaciones legacy.

- **Curva de aprendizaje:**
  Para desarrolladores acostumbrados a escribir código PHP de forma secuencial, el uso de eventos, bucles de eventos (event loops), callbacks y promesas puede resultar complejo al principio. Requiere entender conceptos de asincronía y flujo de ejecución no lineal, lo cual representa un cambio de mentalidad significativo.

- **Limitaciones del entorno de ejecución:**
  Muchas aplicaciones PHP se ejecutan en servidores compartidos con configuraciones limitadas, donde no es posible mantener procesos en ejecución continua (como los que requiere ReactPHP). Esto limita las posibilidades de ejecutar aplicaciones event-driven en infraestructuras tradicionales y obliga a optar por soluciones más avanzadas como servidores dedicados o contenedores.

- **Depuración compleja:**
  El flujo de una aplicación event-driven no sigue una línea recta, sino que salta entre callbacks y manejadores de eventos. Esto puede dificultar la localización de errores o el seguimiento de problemas, especialmente si los errores no están manejados correctamente. Además, la asincronía puede causar errores intermitentes que son difíciles de reproducir.

Mejores prácticas y consideraciones de seguridad:

1. **Modularización:**
   Al usar eventos, es importante diseñar el sistema de forma que cada evento y su listener cumplan funciones específicas y bien separadas. Esto mejora el mantenimiento del código y permite que se puedan añadir o quitar funcionalidades sin afectar otras partes del sistema.

2. **Control de errores asíncronos:**
   En un entorno asincrónico, los errores pueden no aparecer inmediatamente o ser silenciados si no se manejan bien. Es fundamental capturar y registrar adecuadamente los errores dentro de los callbacks o promesas, y usar herramientas de monitoreo para detectar comportamientos anómalos.

3. **Validación de datos en listeners:**
   Al emitir eventos, se pueden transmitir datos sensibles o inconsistentes. Por ello, cada listener debe validar los datos que recibe, incluso si ya fueron validados antes, ya que la fuente puede cambiar o no confiarse completamente.

4. **Límites de recursos:**
   Un servidor que corre continuamente y maneja muchas conexiones puede consumir

demasiados recursos si no se limita adecuadamente. Es recomendable establecer límites de tiempo de ejecución, uso de memoria y número de conexiones activas para evitar cuellos de botella o fallos del servidor.

5. **Uso de librerías probadas:**
   Implementar un sistema de eventos desde cero puede ser riesgoso y propenso a errores. Es mejor apoyarse en librerías maduras como ReactPHP, Amp o Symfony EventDispatcher, que ya han sido probadas y cuentan con soporte de la comunidad.

Conclusiones y Futuras Direcciones

La adopción del paradigma event-driven en PHP tiene profundas implicaciones en la forma en que se diseñan, desarrollan y despliegan aplicaciones web modernas. Este modelo rompe con el enfoque tradicional secuencial y sin estado que históricamente ha caracterizado al lenguaje, promoviendo arquitecturas más reactivas y eficientes.

Entre las principales implicaciones se encuentran:

- **Transformación del modelo de ejecución en PHP:** Se vislumbra una transición progresiva hacia servidores persistentes, que eliminan la necesidad de reiniciar el entorno de ejecución en cada solicitud, lo que mejora sustancialmente el rendimiento en aplicaciones de alta concurrencia.
- **Incremento en el desarrollo de aplicaciones en tiempo real:** La facilidad para implementar comunicaciones bidireccionales mediante WebSockets u otros canales de eventos permite a los desarrolladores construir experiencias más interactivas, como chats, juegos en línea, paneles de control dinámicos y notificaciones instantáneas.
- **Replanteamiento de la arquitectura de software:** Se favorece el uso de patrones modernos como la arquitectura orientada a eventos, Event Sourcing o CQRS, lo cual lleva a una mayor modularidad, escalabilidad y facilidad de mantenimiento de las aplicaciones.
- **Potencial para competir con otras plataformas:** El enfoque event-driven puede posicionar a PHP en terrenos tradicionalmente dominados por plataformas como Node.js o Go, permitiendo construir soluciones equivalentes en rendimiento sin abandonar el ecosistema PHP.

Futuras Líneas de Investigación

Dado el impacto emergente del paradigma event-driven en PHP, se identifican diversas áreas que pueden ser objeto de investigación futura:

1. **Evolución del soporte asincrónico nativo en el núcleo de PHP:**
   Analizar la viabilidad de introducir mecanismos como async/await, promesas o tareas concurrentes directamente en el lenguaje, de forma similar a lo realizado en otros lenguajes modernos.
2. **Desarrollo de frameworks PHP orientados exclusivamente a eventos:**
   Investigar la creación de nuevos frameworks o la adaptación de los existentes para soportar aplicaciones event-driven desde su concepción, proporcionando abstracciones más intuitivas para el manejo de eventos, listeners y bucles de eventos.

3. **Integración del modelo event-driven en entornos heterogéneos:**
   Estudiar cómo las aplicaciones PHP pueden integrarse en arquitecturas distribuidas donde interactúan con servicios desarrollados en otros lenguajes, mediante brokers de eventos como Apache Kafka, Redis o RabbitMQ.
4. **Aplicaciones del paradigma en la educación tecnológica:**
   Explorar cómo este enfoque puede utilizarse en plataformas educativas interactivas, sistemas de respuesta en tiempo real o simuladores colaborativos, aportando nuevas formas de interacción y aprendizaje.
5. **Seguridad y auditoría en sistemas basados en eventos:**
   Investigar mecanismos para asegurar la integridad y trazabilidad de los eventos en sistemas críticos, así como la detección y prevención de comportamientos maliciosos o anómalos en tiempo real.
6. **Impacto del event-driven en la eficiencia energética y uso de recursos:**
   Evaluar cómo el modelo de ejecución persistente y no bloqueante puede contribuir al ahorro de recursos computacionales y energéticos, especialmente en entornos de gran escala o en la nube.

Proyecto: nuestro proyecto consiste en que la ferretería cerna necesita un sitio web dinámico se utilizará ReactPHP para manejar múltiples solicitudes simultáneas de manera no bloqueante. el sitio web tendrá varias páginas que responden a diferentes tipos de solicitudes páginas estáticas, lectura de archivos, consultas a bases de datos y todo será gestionado a través del servidor web basado en eventos.

Link del repositorio de GitHub: https://github.com/DiegoGalanCG20/DSS_investigacion2.git

Bibliografía

[1] W. R. Stevens, *UNIX Network Programming, Volume 1: The Sockets Networking API*, 3rd ed. Addison-Wesley, 2018.
[2] B. Goetz et al., *Java Concurrency in Practice*. Addison-Wesley, 2006.
[3] C. Richardson, *Microservices Patterns: With Examples in Java*. Manning Publications, 2019.
[4] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. Pearson, 2021.

[5] Node.js Foundation, "Node.js Event Loop Documentation," 2023. [Online]. Available: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick
[6] Swoole Team, "Swoole Coroutine and Event Loop," 2023. [Online]. Available: https://www.swoole.co.uk/docs
[7] Gunicorn Documentation, "Worker Types and Performance," 2023. [Online]. Available: https://docs.gunicorn.org/en/stable/design.html

[8] V. Leis et al., "Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age," in *Proc. ACM SIGMOD*, 2015, pp. 743–758.
[9] Microsoft Research, "Memory Management in Event-Driven Systems," 2020. [Online]. Available: https://www.microsoft.com/en-us/research/

[10] TechEmpower, "Web Framework Benchmarks," 2023. [Online]. Available: https://www.techempower.com/benchmarks/
[11] Datadog, "2023 State of Serverless Report," 2023. [Online]. Available: https://www.datadoghq.com/state-of-serverless/

[12] WhatsApp Engineering, "Scaling to 1 Billion Users," 2017. [Online]. Available: https://www.whatsapp.com/scale/
[13] Instagram Engineering, "Migrating Instagram to Python 3," 2017. [Online]. Available: https://instagram-engineering.com
[14] Walmart Labs Tech Blog, "Why We Moved from Java to Node.js," 2020. [Online]. Available: https://medium.com/walmartglobaltech

[15] AWS, "Serverless Architectures with Lambda," 2023. [Online]. Available: https://aws.amazon.com/lambda/
[16] WebAssembly Working Group, "Wasm for Server-Side Applications," 2023. [Online]. Available: https://webassembly.org/
[17] Google Cloud Architecture Center, "Cost Optimization for Concurrent Workloads," 2023. [Online]. Available: https://cloud.google.com/architecture