Design and implementation of a performance scheme for an e-learning platform

Yaneth Moreno *,[a], Gustavo Mejía [b], Francisco Hidrobo [c]

[a] CEMISID, Facultad de Ingeniería, Universidad de Los Andes. Mérida, Venezuela.
[b] Departamento de Computación, Facultad de Ingeniería, Universidad de Los Andes. Mérida, Venezuela.
[c] Departamento de Computación Escuela de Ciencias Matemáticas y Computacionales, Universidad de Tecnología Experimental Yachay Hacienda San José s/n Proyecto Yachay. San Miguel de Urcuquí, Imbabura, Ecuador.

https://doi.org/10.54139/revinguc.v29i2.281

Abstract.- This article proposes the implementation of a transactional scheme for the optimization of access to digital resources (documents, multimedia files, among others) within an e-learning platform, using Moodle as a case study. The transactional schema consists of an extension created to act as an intermediary between the user interface of the e-learning platform and the libraries that manage the storage of data and resources. Additionally, request recommendations on the busiest assets on the platform from an external system. On the other hand, it implements an in-memory database, using radios, in which the most downloaded files by users are stored.

Keywords: Storage System; Transactional Scheme; VLE; Moodle.

Diseño e implementación de un esquema de rendimiento para una plataforma e-learning

Resumen.- En este artículo se propone la implementación de un esquema transaccional para la optimización del acceso a recursos digitales (documentos, archivos multimedia, entre otros) dentro de una plataforma e-learning, utilizando Moodle como caso de estudio. El esquema transaccional consta de una extensión creada con el fin de intermediar entre la interfaz de usuario de la plataforma e-learning y las bibliotecas que gestionan el almacenamiento de datos y recursos. Además, solicita recomendaciones sobre los recursos más concurridos de la plataforma a un sistema externo. Por otro lado, implementa una base de datos en memoria, empleando Redis, en la que se almacenan los archivos más descargados por los usuarios.

Palabras clave: Sistema de Almacenamiento; Esquema transaccional; EVA; Moodle.

Recibido: 21 de noviembre, 2022.
Aceptado: 16 de enero, 2023.

1. Introducción

La evolución de las tecnologías de información y comunicación ha traído cambios significativos para el hombre. Dichos cambios se han reflejado en distintos sectores, destacando de manera importante en la educación. En los últimos años, las plataformas dedicadas a la formación de profesionales a distancia han ganado un cuantioso número de usuarios, en distintas carreras y áreas de formación. La finalidad ha sido proveer oportunidades a quienes por una razón u otra no pueden acceder a una educación presencial, o que necesitan complementar dicha formación usando recursos virtuales. Por otro lado, la pandemia del COVID-19 ha dejado latente que los estudios a distancia y la educación virtual ya no se ven solo como una alternativa [1], sino como un complemento y posiblemente como un esquema de sustitución a la educación tradicional (presencial).

Entre las diferentes plataformas de aprendizaje electrónico que hoy existen, podemos resaltar el proyecto de software libre Moodle. Esta se define como una plataforma de aprendizaje diseñada para proporcionar a educadores, administradores y estudiantes un sistema integrado único, robusto y seguro para crear ambientes de aprendizaje

* Autor para correspondencia:
*Correo-e:*yanethmoreno2002@gmail.com (Y. Moreno)

personalizados [2].

No obstante, a pesar del apoyo y soporte que hay detrás del mundo del e-learning, estas plataformas no escapan de las problemáticas comunes de los sistemas web en general. La gran cantidad de usuarios que acceden diariamente a distintos cursos en línea y de manera simultánea, pueden acarrear problemas en los tiempos de respuesta, si no se cuenta con los recursos necesarios para adaptar la infraestructura a los nuevos requerimientos de los sistemas. Así, surge la necesidad de aumentar la capacidad de respuesta y la eficiencia de las aplicaciones web, entre ellas las plataformas e-learning. La idea es enriquecer la calidad en cuanto a la experiencia de uso, es decir, que el servicio satisfaga las expectativas y necesidades del usuario, y, por otro lado, mejorar la calidad de servicio, referida a la medida del rendimiento del sistema.

El rendimiento de una aplicación web es importante, pero en una plataforma de formación es crucial. La percepción de los estudiantes sobre la calidad de los cursos puede verse deteriorada si durante su formación la plataforma responde de forma lenta o inesperada. Por ende, se requieren Entornos Virtuales de Aprendizaje (EVA) más eficientes en la gestión de los requerimientos de usuarios, cada vez más exigentes, y que ellos perciban u observen que su productividad no está siendo afectada. En este trabajo, se propone mejorar las prestaciones en cuanto a la gestión de almacenamiento. Para ello, se desarrolló e implementó un esquema de rendimiento apropiado que, integrado a la plataforma, permite acceder con menos retraso a los archivos solicitados por los usuarios.

2. Metodología

En este trabajo se aplicó un Desarrollo Basado en Funcionalidades (FDD, por sus siglas en inglés) [3], puesto que se trata de una metodología ágil que se fundamenta en la calidad y el monitoreo constante de cada proceso. Se enfoca en iteraciones cortas, que permiten avances tangibles en un período corto de tiempo. La misma consta de cinco procesos: desarrollo de un modelo global, creación una lista de funcionalidades, planificación por funcionalidad, diseño y desarrollo de cada funcionalidad y por último, la implementación de las funcionalidades.

3. Estado del arte

Para crear un sistema de rendimiento integrado en una plataforma e-learning, que contribuya a optimizar el sistema gestor de almacenamiento, fue necesario realizar un estudio tanto de algunas implementaciones, como de distintos trabajos relacionados con el tema planteado. Las primeras revisiones se hicieron a algunas extensiones (*plugins*) de la plataforma Moodle. Las extensiones existentes cubren algunas necesidades de la plataforma o brindan funcionalidades adicionales a esta. Las mismas pueden ser instaladas de forma manual o a través de la página oficial de Moodle (moodle.org).

Específicamente, la categoría que fue de apoyo para este trabajo, es la de los *plugins* que integran memoria *cache*, entre los cuales resalta Alternative PHP Cache (APC) (Hemelryk, 2014. https://moodle.org/plugins/cachestore_apc), el cual proporciona una *cache* de tamaño limitado, pero de excelente desempeño, destinada al almacenamiento de datos de aplicaciones persistentes de PHP; y Memcache Cluster (Merrill, 2014. https://moodle.org/plugins/cachestore_memcachecluster) que implementa una versión modificada de la memoria cache estándar de Moodle, que permite mantener varios almacenamientos *memcache* sincronizados entre sí.

Luego, se realizó un estudio de la biblioteca AdoDB, una capa de abstracción de base de datos muy conocida, rápida y fácil de usar para PHP y que sirve para facilitar la comunicación entre Moodle y su base de datos. Con esto se buscaba comprender los procesos transaccionales entre la plataforma y los datos. En uno de los trabajos estudiados se empleó el lenguaje PHP en conjunto con la biblioteca AdoDB para generar interfaces de usuario web dinámicamente, evaluando la posibilidad de utilizar metadatos almacenados en bases de datos para desarrollar elementos de interfaz de usuario [4]. También se analizó un *framework* para PHP basado en

el modelo de 3 capas con el fin de identificar y separar la aplicación final en diferentes capas, que faciliten su construcción y mantenimiento. Este enfoque integra diferentes tecnologías y patrones de diseño con el fin de proporcionar una herramienta eficaz que respaldo a la comunidad en la creación de aplicaciones web con PHP, entre estas, se integra la biblioteca AdoDB en la capa de acceso a datos para gestionar la comunicación entre la aplicación y la base de datos relacional [4]. Por otra parte, se revisó el diseño e implementación de una aplicación móvil para la intranet, propuesto en la Universidad Politécnica de Madrid, basada en el framework **Cordova** y tecnologías web [5]. La idea de dicho trabajo, fue proponer la programación híbrida usando el patrón Modelo-Vista-Controlador (MVC) como una forma de programación para sustituir el desarrollo ordinario para plataformas nativas, evitando el desarrollo en paralelo para cada sistema operativo móvil, como Android o iOS, y brindando una mejor implementación. Y el propósito de utilizar la biblioteca AdoDB en este trabajo fue, principalmente, evitar el mantenimiento extra en el caso de migraciones de la base de datos o cambios en los controladores de esta. Además, la implementación de AdoDB brinda a los desarrolladores la facilidad de aplicar el mismo código para acceder a una amplia gama de bases de datos.

4. Análisis de la plataforma Moodle

Moodle es una plataforma web de aprendizaje colectivo, de código abierto, diseñada para soportar tanto la enseñanza como el aprendizaje guiado por la pedagogía de constructivismo social, puesto que proporciona un conjunto de poderosas herramientas centradas en el estudiante y ambientes de aprendizaje colaborativo [2].

El proyecto Moodle impulsa decenas de miles de ambientes de aprendizaje globalmente, con más de 79 millones de usuarios, entre académicos y empresariales, que la convierten en la plataforma de aprendizaje más ampliamente usada del mundo [6].

En este trabajo es utilizada como caso de estudio la versión 3.3 de Moodle, para la implementación de un esquema transaccional de datos.

4.1. Extensiones de Moodle

En informática, un complemento o plug-in es una aplicación que se relaciona con otra para agregarle una función nueva, y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la interfaz de programación de aplicaciones (API por sus siglás en inglés).

Moodle cuenta con un amplio directorio de extensiones que permiten agregar características y funcionalidades extras, como por ejemplo, nuevas actividades, nuevos tipos de preguntas para exámenes, nuevos reportes, integraciones con otros sistemas y muchas más.

Entre las extensiones de Moodle de interés para este trabajo se encuentran algunas relacionadas con el manejo de memoria *cache*, como por ejemplo, Memcached, MongoDB, APC user cache (APCu), XCache y Redis. Una *cache* es un componente que almacena datos para que las solicitudes futuras de esos datos se puedan atender con mayor rapidez.

4.2. Estructura de una extensión

Una extensión en Moodle posee los componentes que se describen a continuación:

version.php: contiene la meta información sobre el plugin, por ejemplo, la versión de este.
settings.php: archivo opcional que contiene el formulario con las opciones generales del plugin.
index.php: sirve para mostrar todas las instancias de una actividad en un curso, es decir, una lista con todas las instancias del mismo plugin.
view.php: esta es la página que muestra una instancia de la actividad.
lib.php: biblioteca de funciones del plugin. En este archivo se implementarán todas sus funciones y procedimientos.
mod form.php: formulario para crear o modificar una instancia de la actividad.
lang/: almacenar los archivos de idioma del plugin. Este debe contener los archivos de idioma con las cadenas de texto necesarias por el plugin en inglés

y sus traducciones a los idiomas de los usuarios finales.

db/: directorio donde se almacenan los archivos con las tablas de las bases de datos necesarias.

access.php: archivo opcional que contiene los permisos del plugin. Los permisos no son obligatorios, pero sí recomendables para garantizar el control de acceso de los usuarios a las funcionalidades específicas.

install.xml: archivo que describe la estructura de las tablas del plugin.

upgrade.php: código de actualización, aquí es donde se hacen las alteraciones de las tablas, si las hay, entre versiones.

5. Planificación y diseño de funcionalidades

Como primer paso, se desarrolló un modelo global de la solución a implementar, se definió una lista de funcionalidades con el fin de satisfacer dicho modelo y seguidamente se planificó el desarrollo de las mismas.

5.1. Desarrollo de un modelo global

El modelo global consta de la integración de un sistema recomendador externo a Moodle y una extensión que maneje las transacciones entre dicho sistema, la memoria *cache* implementada con Redis, las bibliotecas de la plataforma y la interfaz de usuario.

La extensión desarrollada se ubica entre la interfaz de usuario y las bibliotecas de Moodle que se encargan del manejo de los datos, como se observa en la Figura 1, funcionando como una capa intermedia o middleware que recibe tanto las consultas de cada usuario como las respuestas desde el servidor.

A su vez, la extensión implementada se comunica con el servidor de Redis y con el recomendador externo. Se utilizó Redis por ser un motor de base de datos que hace uso de la memoria principal del computador para alojar datos en un esquema clave-valor, que permite almacenar archivos binarios que no excedan un tamaño de 512 MB, y por su rápida integración con el lenguaje de programación PHP. Su función es la de mantener copias de los archivos a los cuales los usuarios acceden con mayor frecuencia en la plataforma con el fin de proveer una respuesta más rápida a sus solicitudes.

El recomendador externo es el encargado de decidir y notificar a la extensión qué archivos deben ser copiados a la memoria *cache*. La extensión, por su parte, solicitará al recomendador externo, cada vez que sea necesario, un listado actualizado de los archivos, que este determine, deban permanecer en la memoria *cache* y creará copias de estos en la misma, así como proveer al recomendador un listado de aciertos y errores que pueda usar para futuras recomendaciones.

El fin de esta memoria *cache* es almacenar los archivos más concurridos y debe ser actualizada cada vez que se produzca una nueva recomendación. El plugin se encarga de remover los archivos que ya no sean necesarios y cubrir este espacio con nuevos recursos.

El componente no solo debe encargarse de dar respuesta a las solicitudes cuando el recurso solicitado se encuentre en la memoria *cache*, sino que también debe responder cuando este no se encuentre a primera mano, buscándolo en el servidor de archivos y enviándolo al usuario que lo ha solicitado.

5.2. Construcción de una lista de funcionalidades

Una funcionalidad es una pequeña utilidad valorada por el cliente, que se expresa en forma de acción - resultado - objeto. Por ejemplo, “Calcular el total de una venta”, “Validar la contraseña de un usuario” y “Autorizar la transacción de venta de un cliente” [8]. Acorde a lo descrito en la sección anterior, en la cual se presentó un modelo global de extensión propuesta, la lista de funcionalidades que se estableció es la siguiente:

- Recomendaciones
  - Solicitar recomendaciones al recomendador externo (HTTP Request).
  - Enviar métricas de aciertos y errores al recomendador externo (HTTP Request).
- Memoria *cache*
  - Limpiar archivos innecesarios de la memoria *cache*, cada vez que se produzca una nueva recomendación.

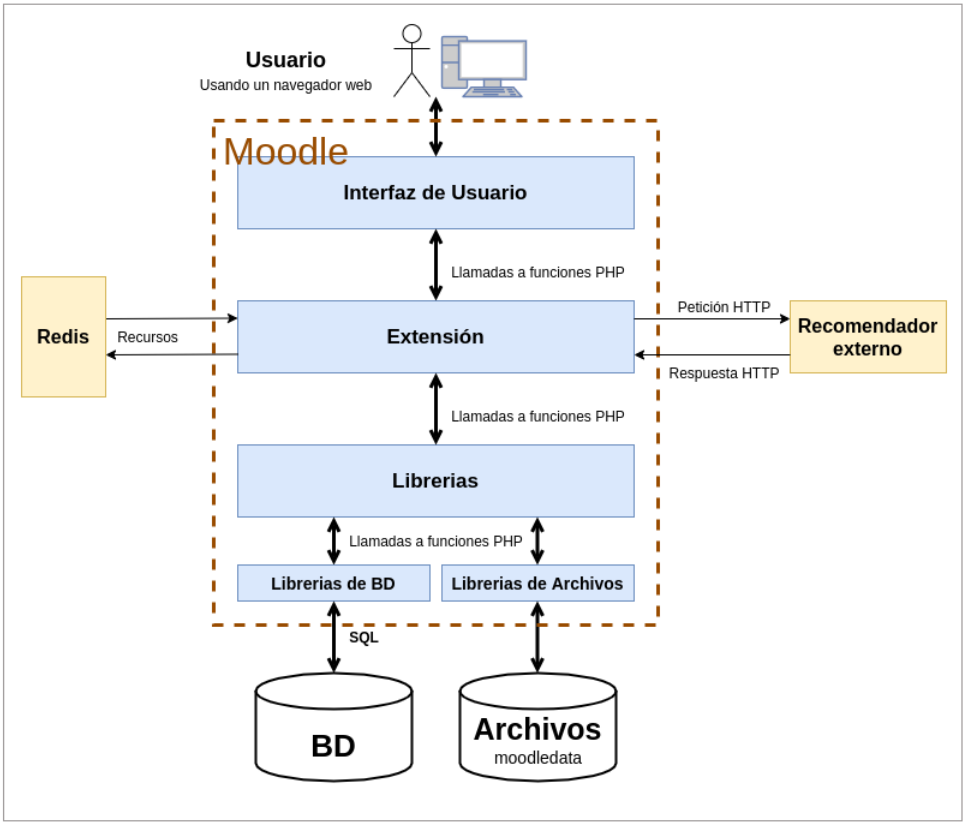

Figura 1: Ubicación de la extensión en la arquitectura Moodle (modificado de [7])

- Copiar archivos nuevos a la memoria *cache* de acuerdo a la última recomendación producida.

- Descargas
  - Descargar archivos solicitados por los usuarios desde la memoria *cache*, en caso de encontrarse en la misma.
  - Descargar archivos solicitados por los usuarios desde el almacenamiento, en caso de no encontrarse en la memoria *cache*.

5.3. *Diseño de funcionalidades*

El diseño de los sets de funcionalidades se realizó al comienzo de cada iteración en la cual correspondía su desarrollo, como se estableció en la sección anterior.

Recomendaciones

La solicitud de recomendaciones y el envío de métricas se deben efectuar de forma automática, cada vez que sea necesario. Por ende, la extensión hace uso de las tareas sincronizadas de Moodle. Una tarea es una unidad de trabajo que debe ser efectuada en un tiempo determinado, son especialmente útiles para ejecutar una tarea de mantenimiento en un horario regular.

En la Figura 1 también se puede observar la interacción entre la extensión y el recomendador externo. Las recomendaciones y métricas se recibirán y enviarán como solicitudes HTTP, en formato JSON.

El diagrama de secuencia mostrado en la Figura 2 corresponde al proceso de solicitud de recomendaciones y envío de métricas para la ejecución de la tarea programada, encargada de comunicarse con el recomendador externo.

Memoria cache

La memoria *cache* implementada en este trabajo tiene la función de almacenar los archivos más solicitados por los usuarios de la plataforma

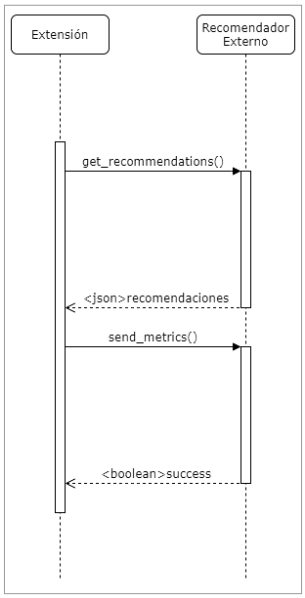

Figura 2: Diagrama de secuencia de la interacción extensión-recomendador

Moodle, en un cierto lapso de tiempo.

Como se observa en la Figura 3, luego de recibir un listado de recomendaciones, desde el recomendador externo mencionado en la sección anterior, nuestro componente debe limpiar de la memoria *cache* los archivos que no se contemplen en este listado antes de proceder con el copiado.

Posteriormente, el componente busca uno por uno los archivos del listado en el almacenamiento de archivos. Para cada caso se verifica que el recurso no se encuentre almacenado ya en la memoria *cache* y, de no ser así, se verifica si el espacio disponible es suficiente para agregarlo a la memoria, considerando el límite establecido. Si el almacenamiento disponible es mayor al tamaño del archivo, se procede a copiarlo. Para este proyecto se fijó un límite de aproximadamente de 800 MB ya que se contó con un almacenamiento físico de 2 GB, este valor se establece como un parámetro de configuración.

Antes de continuar con el siguiente archivo se debe actualizar la memoria usada. De quedar entradas sin revisar en el listado, el componente buscará el siguiente archivo y realizará el proceso. De no poseer espacio suficiente, se procede con el siguiente recurso del listado hasta que este sea cubierto en su totalidad o la memoria se haya utilizado por completo.

Descargas

Para integrar estas funcionalidades al flujo normal de los procesos de Moodle es necesario adentrarse en su código fuente, específicamente en la rutina encargada de gestionar todas las descargas que los usuarios solicitan. La mayoría de las actividades que involucran la descarga de archivos y reproducción de recursos multimedia hacen uso de la rutina *pluginfile.php* que Moodle posee en su código fuente. Esta rutina sirve de interfaz, tomando directamente la solicitud de los usuarios para crear el enlace de descarga del recurso solicitado. Por ende, es allí en donde se hace la inserción del *plugin*.

Sin embargo, ya que no siempre se servirán archivos desde la memoria *cache* administrada por nuestra extensión, solo se realizó una bifurcación, tal como se puede apreciar en la Figura 4, en la que luego de obtener los argumentos del archivo solicitado se verifica si este existe o no en *cache*. Si el resultado de esta verificación es satisfactorio, se procede a buscar el archivo en la memoria, si, por el contrario, se determina que el archivo no se encuentra en la misma, el proceso continúa de la misma manera que lo haría si el *plugin* diseñado no estuviera implementado.

Luego de determinar el método a utilizar se deben actualizar las métricas de aciertos y errores. En caso de que el archivo sea encontrado en la memoria *cache*, la cantidad de aciertos de las recomendaciones sumarían uno. Por el contrario, si el archivo debiera obtenerse desde el almacenamiento, representaría un desacierto y sumaría uno a la cuenta de fallos. Estas métricas son escritas en una tabla de la base de datos creada con este fin llamada *download optimizer metrics*, y pueden ser usados por el recomendador para alimentar los procesos de reentrenamiento y adaptación.

6. Implementación y discusión de los resultados

En esta sección se detalla el proceso de implementación, seguido de las pruebas que se

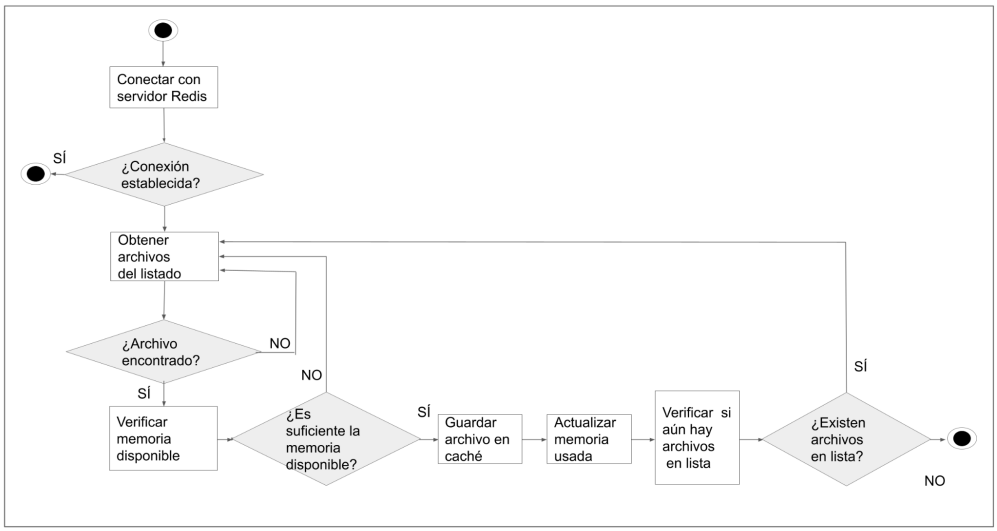

Figura 3: Diagrama de actividad: Copiado de un archivo a la memoria *cache*

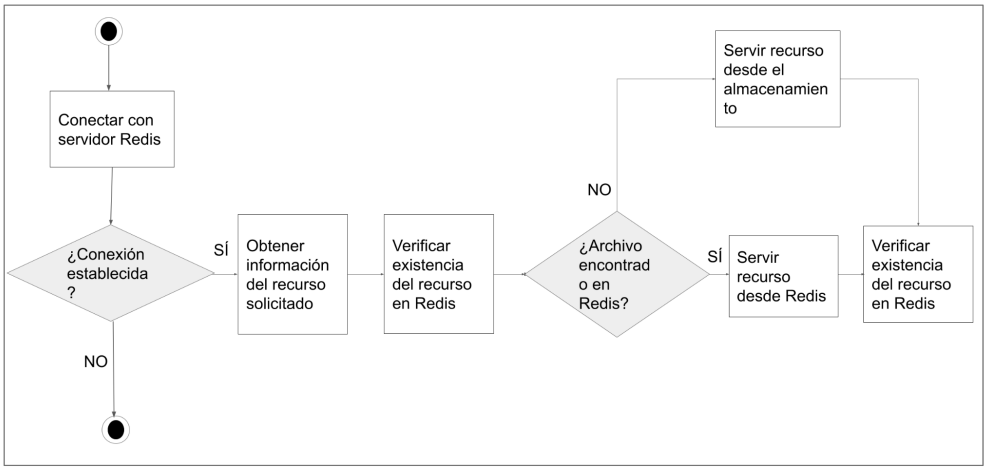

Figura 4: Diagrama de actividad: Descarga de archivos

ejecutaron para comprobar el funcionamiento y rendimiento del esquema transaccional propuesto. Para estas pruebas se empleó un equipo que contaba con un procesador Intel Core 2 Duo, un disco duro mecánico de 256 GB de almacenamiento y una memoria de acceso aleatorio DDR de 2 GB. En cuanto al software, la implementación corrió sobre el Sistema Operativo Linux, específicamente Debian 9 y se utilizó la versión 3.3 de Moodle.

6.1. Implementación de funcionalidades

El último proceso de la metodología trata de la implementación de las funcionalidades y se aplicó de forma iterativa en conjunto con el proceso de diseño. Antes de cada iteración se seleccionaron los sets de funcionalidades que serían diseñados e implementados en la misma.

Recomendaciones

Se requería que la solicitud de recomendaciones y el envío de métricas al recomendador externo se hiciera de forma automática en intervalos de tiempo que el administrador de la plataforma decidiera sean adecuados. Para conseguir esto, se implementaron estas funcionalidades en una tarea sincronizada de Moodle. Las tareas sincronizadas son ejecutadas a través del proceso Cron de Moodle que es un script PHP contenido en el código fuente

que debe ejecutarse regularmente en segundo plano y se encarga de ejecutar las tareas sincronizadas en sus intervalos agendados.

El programa cron.php de Moodle debe ser invocado regularmente, esto con el fin de que las tareas sincronizadas se ejecuten correctamente en los intervalos establecidos. Ya que PHP permite ejecutar programas desde la consola de comandos de Linux, se configuró el Crontab de Linux para que ejecute cron.php cada minuto. Cron es un administrador regular de procesos en segundo plano que ejecuta procesos o rutinas a intervalos regulares como se especifican en el archivo crontab.

En primer lugar, se invoca a la rutina *check metrics availability* que se encarga de verificar que se encuentren las entradas necesarias en la base de datos y de no encontrarse, se crean. Esto es útil como seguridad para evitar errores que se puedan presentar al momento de escribir los aciertos y errores en la tabla.

Posteriormente, se invoca la rutina *send metrics* que envía las métricas capturadas al recomendador. Esta función se encarga de recuperar de la base de datos los valores capturados de aciertos y errores para el último listado de recomendaciones y enviarlos en una petición HTTP de tipo POST usando la biblioteca cURL de PHP. Una vez finalizado el envío de la petición se restablecen los valores de la tabla a cero.

La función get recommendations se encarga de solicitar al recomendador una lista de recomendaciones actualizada. Para conseguir esto también se utiliza la biblioteca cURL, pero en este caso se envía una petición de tipo GET. Esta rutina retorna un arreglo que contiene los identificadores de los archivos más concurridos en cada una de sus posiciones. En caso de que ocurra un fallo, se retorna un mensaje de error.

Memoria cache

En este proyecto se implementó una memoria *cache* empleando el motor de base de datos en memoria Redis, el cual es posible manejar desde plugin implementado a través de la biblioteca de PHP PhpRedis 7, que proporciona una API para comunicarse con el almacenamiento clave-valor empleando código PHP.

Descargas

La función *clear cache* se encarga de remover de la memoria *cache* los archivos que ya no son necesarios, es decir, que no se encuentran en el listado recibido. Esta rutina toma un arreglo conteniendo el nuevo listado de recomendaciones y se conecta con Redis, usando la instrucción *connect* de la biblioteca PhpRedis. Esta última recibe como argumentos la dirección y el puerto del servidor Redis y obtiene todas las claves contenidas en la memoria con la instrucción *keys*, estas claves corresponden a los identificadores de los archivos como se encuentran registrados en la base de datos de Moodle. Al comparar los identificadores contenidos en *cache* con los que se encuentran en el nuevo listado de recomendaciones, se obtiene un arreglo con las claves innecesarias. La instrucción *"del"* toma como argumento el arreglo de las claves a eliminar y las remueve del servidor. De ser removidos todos los archivos innecesarios, la función retorna el valor booleano TRUE, en caso contrario, Muestra un mensaje de error en la consola y retorna el valor booleano FALSE.

La función *retrieve files* recibe como argumento el arreglo de recomendaciones recibido desde el recomendador y se encarga de copiar los archivos desde el almacenamiento de Moodle a la memoria *cache*. Ya que esta función también debe velar que el límite para la *cache* establecido por el administrador no se exceda, se usa la instrucción *info*, en este caso con el argumento MEMORY para obtener el espacio utilizado actualmente.

La función *get cache limit* obtiene el tamaño de la memoria RAM del computador y retorna el límite de la memoria *cache* en función a la anterior. Para este proyecto se estimó emplear la tercera parte del total de la memoria física del computador (aproximadamente 800 MB en este caso) pero este valor puede ser modificado a conveniencia del administrador.

La rutina *retrieve files* itera sobre cada identificador contenido en el arreglo de recomendaciones recibido, y para cada caso usa la API de archivos (File API 8) de Moodle, que es la interfaz de programación de aplicaciones que Moodle provee para manejar los archivos en la plataforma, con el fin de obtener los metadatos del archivo

en cuestión, junto con su contenido. Si no existe espacio disponible suficiente para almacenar este archivo, se procede con el siguiente y así sucesivamente. Sí, por el contrario, el espacio disponible es suficiente, se utiliza la función *redis save file*, que recibe como argumentos el identificador del archivo y el contenido del mismo y lo almacena en la memoria *cache* utilizando la instrucción *set* de PhpRedis. De ser satisfactorio el proceso, se actualiza la variable que contiene el valor de la memoria utilizada y se procede con el siguiente archivo. Por el contrario, si el proceso falla, se retorna un mensaje de error a la consola y se procede de igual forma con el siguiente archivo.

6.2. *Pruebas*

6.2.1. *Uso de las instrucciones var dump( ) y echo para verificar el estado del proceso*

La instrucción *var dump( )* de PHP muestra información estructurada sobre una o más expresiones, incluyendo su tipo y valor, esta se implementó como herramienta para conocer información de utilidad en la ejecución del proceso de limpiado y llenado de la memoria *cache*, como por ejemplo el espacio disponible al almacenar o remover un elemento de la misma.

Por otro lado, *echo* se define como un constructor del lenguaje, su función es imprimir un texto dado. Su implementación se aplicó para imprimir mensajes que explicaran de forma rápida y clara lo que estaba ocurriendo en dicho momento, un ejemplo serían los mensajes de éxito y/o error en algún paso de la ejecución.

6.2.2. *Pruebas de rendimiento*

Con el fin de conocer el rendimiento de nuestra extensión y realizar las respectivas comparaciones con el flujo normal que provee la plataforma por omisión, se implementó un sistema de logs que son plasmados en la base de datos en una tabla que lleva por nombre *download optimizer logs*. La información capturada en esta corresponde al tiempo en microsegundos que se toma la plataforma en servir un archivo especificado con su propio identificador, ya sea que fuese servido desde la memoria *cache* o directamente desde el almacenamiento.

6.3. *Resultados*

A continuación, se muestran los resultados obtenidos, tabulados para los tres casos. Como se puede apreciar en la Tabla 1, se utilizaron archivos PDF, FLAC y MP4 de tamaños variados con el fin de probar la mayor cantidad posible de comportamientos de la plataforma.

Tabla 1: Tamaño de archivos por formato

	Tipos de archivos		
Archivo	**PDF** (MB)	**FLAC** (MB)	**MP4** (MB)
1	12.74	23.20	23.20
2	126.12	125.92	125.92
3	395.89	329.74	329.74

En las Tablas 2, 2 y 4 se muestra una comparación entre las medias y varianzas de los tiempos de respuesta tanto del servidor como de la *cache*.

En general, se puede observar que en las pruebas para los distintos tipos de archivo se percibió una mejora. También se puede apreciar que el rendimiento en los archivos más pequeños fue mejor que en los archivos con mayor tamaño, alcanzando mejoras de 91,28 % en archivos PDF, 72,19 % en archivos de audio FLAC y 91,94 % en archivos de video MP4.

Por otro lado, el rendimiento en general de la plataforma para servir archivos de audio y video se reduce con mayor proporción en comparación con archivos de texto PDF a medida que aumenta el tamaño de los mismos.

6.4. *Conclusiones*

Mediante este trabajo se logró la implementación de un esquema transaccional para mejorar los tiempos de respuesta de la plataforma Moodle al momento de servir recursos a los usuarios, dicho esquema se llevó a cabo mediante una extensión en la plataforma. Para llevar esto a cabo, se aplicó una metodología de desarrollo basada en funcionalidades, que permitió obtener en principio un modelo global óptimo.

Luego, se construyó una lista de características enfocada en satisfacer las necesidades de dicho modelo, con el fin de cumplir con la implementación planteada. Posteriormente, se realizó la

Tabla 2: Medias y varianzas de tiempos de ejecución registrados para archivos PDF

	Cache (ms)	Var (cache)	Servidor (ms)	Var (servidor)
Archivo 1	0,0008379	0,00000030175	0,0086181	0,00043929530
Archivo 2	0,0427598	0,06056147000	0,0605615	0,00040822452
Archivo 3	0,1173530	0,00030477933	0,1438336	0,04523632832

Tabla 3: Medias y varianzas de tiempos de ejecución registrados para archivos FLAC

	Cache (ms)	Var (cache)	Servidor (ms)	Var (servidor)
Archivo 1	0,0044143	0,00000271154	0,0158703	0,00125392680
Archivo 2	0,4876548	0,00875906785	0,5536980	0,00875906785
Archivo 3	0,6993443	0,33901860605	0,7014644	0,00875906785

Tabla 4: Medias y varianzas de tiempos de ejecución registrados para archivos MP4

	Cache (ms)	Var (cache)	Servidor (ms)	Var (servidor)
Archivo 1	0,0004151	0,00000009664	0,0006176	0,00024485936
Archivo 2	0,0076583	0,20519410250	0,4544273	0,06293746897
Archivo 3	0,6901814	0,00700439286	0,6903727	0,00700439286

planificación y el diseño de cada funcionalidad por separado, pero manteniendo la integridad del sistema para finalmente realizar la implementación.

Con la extensión implementada, se ejecutaron las pruebas de funcionamiento y rendimiento correspondientes, en las cuales se determinó que el trabajo hecho cumplía con los objetivos establecidos, resultando en una implementación completamente funcional del modelo diseñado.

Con base en los resultados obtenidos, se pudo comprobar que efectivamente la extensión diseñada mejora la experiencia del usuario dentro de la plataforma al momento de acceder a algún recurso de alta demanda, optimizando los tiempos de respuesta de la misma dependiendo de las propiedades de los archivos.

7. Referencias

[1] M. García, "La docencia desde el hogar. Una alternativa necesaria en tiempos del Covid 19," *Revista Polo del Conocimiento*, vol. 5, no. 4, pp. 304–324, 2020. https://doi.org/10.23857/pc.v5i3.1318

[2] G. Barrios, Y. Moreno, and F. Hidrobo, "Entendiendo el funcionamiento de Moodle: un enfoque basado en un marco de modelado," *RISTI-Revista Ibérica de Sistemas e Tecnologias de Informação*, vol. 20, pp. 327–337, 2019.

[3] A. Navarro, J. Fernández, and J. Morales, "Revisión de metodologías ágiles para el desarrollo de software PROSPECTIVA," *Universidad Autónoma del Caribe*, vol. 11, no. 2, pp. 30–39, 2013.

[4] M. Mgheder and M. Ridley, "Automatic generation of web user interfaces in php using database metadata," in *Third International Conference on Internet and Web Applications and Services ICIW'08*, Athens, Greece, 2008, pp. 426–430. https://doi.org/10.1109/ICIW.2008.100

[5] G. García, "Design and implementation of a mobile application based on cordova framework and web technologies for a university intranet," Tesis de Maestría (TFG), ETSI Telecomunicación. Universidad Politécnica de Madrid, Madrid, España, 2017.

[6] M. Rizo, "Aprendizaje con Moodle," *Revista Multi-Ensayos*, vol. 4, no. 8, pp. 18–25, 2018. https://doi.org/10.5377/multiensayos.v4i8.9448

[7] A. Muñoz, R. Delgado, E. Rubio, C. Grilo, and V. Basto-Fernandes, "Forum participation plugin for Moodle: Development and Discussion," *Procedia Computer Science*, vol. 121, pp. 982–989, 2017. https://doi.org/10.1016/j.procs.2017.11.127

[8] F. Anwer, S. Aftab, U. Waheed, and S. Muhammad, "Agile Software Development Models TDD, FDD, DSDM, and Crystal Methods: A Survey," *International Journal of Multidisciplinary Sciences and Engineering*, vol. 8, no. 2, pp. 1–10, 2017.