Facultad de Ingeniería
ISSN: 0121-1129
ISSN: 2357-5328
revista.ingenieria@uptc.edu.co
Universidad Pedagógica y Tecnológica de Colombia
Colombia

Flores-González, Martín; Trejos-Zelaya, Ignacio; Garita, César
Modelado exploratorio del rendimiento y la confiabilidad de software sobre middleware orientado a mensajes
Facultad de Ingeniería, vol. 29, núm. 54, 2020, -Marzo
Universidad Pedagógica y Tecnológica de Colombia
Colombia

DOI: https://doi.org/10.19053/01211129.v29.n54.2020.11764

Disponible en: https://www.redalyc.org/articulo.oa?id=413962511038

Exploratory Modeling of Software Performance on Message-Oriented Middleware

Martín Flores-González; Ignacio Trejos-Zelaya; César Garita

Citación: M. Flores-González, I. Trejos-Zelaya, C. Garita, “Exploratory Modeling of Software Performance on Message-Oriented Middleware,” *Revista Facultad de Ingeniería*, vol. 29 (54), e11764, 2020. https://doi.org/10.19053/01211129.v29.n54.2020.11764

Recibido: Julio 26, 2020; **Aceptado:** Septiembre 14, 2020; **Publicado:** Septiembre 15, 2020

Conflicto de intereses: Los autores declaran no tener conflicto de intereses.

Exploratory Modeling of Software Performance on Message-Oriented Middleware

Martín Flores-González[1]
Ignacio Trejos-Zelaya[2]
César Garita[3]

Abstract

Performance is an important quality attribute in a software system. Software Performance Engineering comprises analysis, design, construction, measurement and validation concerning performance requirements during software development processes. Performance in software systems using message-based communication depends mostly on the Message-Oriented Middleware (MOM). Software architects need to consider MOM's organization, configuration and usage details to get meaningful predictions about the behavior of a software system that uses such platform. When including MOM in software architecture, it is required to foresee the impact of messaging and its underlying infrastructure. Software architects may omit the MOM influence, which could lead to wrong predictions. In this article, we explore MOM's influence through the Palladio Component Model – PCM, a component-based modeling and simulation approach. An application previously modeled with PCM was adapted to include message-oriented communication. Simulations over the model, systematic measurements, and load testing on the adapted application were performed, in order to determine how the changes in the model influenced the

[1] M. Sc. Instituto Tecnológico de Costa Rica (Cartago, Costa Rica). ORCID: 0000-0003-0161-0049
[2] M. Sc. Instituto Tecnológico de Costa Rica (Cartago, Costa Rica). itrejos@tec.ac.cr. ORCID: 0000-0003-4361-8444
[3] Ph. D. Instituto Tecnológico de Costa Rica (Cartago, Costa Rica). cesar@tec.ac.cr. ORCID: 0000-0003-4592-3266

prediction of the application's behavior on performance and reliability. A bottleneck that impacts performance and reliability of the original system was identified. Introducing MOM improved the system's reliability but harmed its performance. Component-based performance simulation revealed significant differences with measurements obtained during the load testing experiments.

Keywords*:* message-oriented middleware; Palladio Component Model; software modeling and simulation; software performance; software performance engineering; software reliability.

Modelado exploratorio del rendimiento y la confiabilidad de software sobre *middleware* orientado a mensajes

Resumen

El rendimiento es un importante atributo de calidad de un sistema de software. La Ingeniería de rendimiento del software comprende las actividades de análisis, diseño, construcción, medición y validación, que atienden los requerimientos de rendimiento a lo largo del proceso de desarrollo de software. En los sistemas de software que utilizan comunicación basada en mensajes, el rendimiento depende en gran medida del *middleware* orientado a mensajes (*Message-Oriented Middleware – MOM*). Los arquitectos de software necesitan considerar su organización, configuración y uso para predecir el comportamiento de un sistema que use tal plataforma. La inclusión de un MOM en una arquitectura de software requiere conocer el impacto de la mensajería y de la infraestructura utilizada. Omitir la influencia del MOM llevaría a la generación de predicciones erróneas. En este artículo se explora tal influencia, mediante el modelado y la simulación basados en componentes, utilizando el enfoque *Palladio Component Model – PCM*. En particular, una aplicación modelada en PCM fue adaptada para incluir comunicación basada en mensajes. Las simulaciones sobre el modelo, mediciones sistemáticas y pruebas de carga sobre la aplicación permitieron determinar cómo cambios introducidos en el modelo influyen en las predicciones del comportamiento de la aplicación en cuanto a rendimiento y confiabilidad. Fue posible identificar un cuello de botella que impacta negativamente el rendimiento y la confiabilidad del sistema

original. La introducción de MOM mejoró la confiabilidad del sistema, a expensas del rendimiento. La simulación del rendimiento basado en componentes reveló diferencias significativas respecto de los experimentos basados en pruebas de carga y mediciones.

Palabras clave: confiabilidad del software; ingeniería de rendimiento de software; middleware orientado a mensajes; modelado y simulación de software; Palladio Component Model; rendimiento del software.

Modelagem exploratória de desempenho e confiabilidade de software em middleware orientado a mensagens

Resumo

O desempenho é um importante atributo de qualidade de um sistema de software. A engenharia de desempenho de software compreende as atividades de análise, projeto, construção, medição e validação, que atendem aos requisitos de desempenho em todo o processo de desenvolvimento de software. Em sistemas de software que usam comunicação baseada em mensagens, o desempenho é altamente dependente do Message-Oriented Middleware (MOM). Os arquitetos de software precisam considerar sua organização, configuração e uso para prever o comportamento de um sistema usando essa plataforma. A inclusão de um MOM em uma arquitetura de software requer o conhecimento do impacto das mensagens e da infraestrutura usada. Omitir a influência do MOM levaria à geração de previsões erradas. Este artigo explora essa influência, por meio de modelagem e simulação baseada em componentes, utilizando a abordagem Palladio Component Model - PCM. Em particular, um aplicativo modelado em PCM foi adaptado para incluir comunicação baseada em mensagens. As simulações no modelo, medições sistemáticas e testes de carga na aplicação permitiram determinar como as alterações introduzidas no modelo influenciam nas previsões do comportamento da aplicação em termos de desempenho e fiabilidade. Foi possível identificar um gargalo que impacta negativamente o desempenho e a confiabilidade do sistema original. A introdução do MOM melhorou a confiabilidade do sistema, em detrimento

do desempenho. A simulação de desempenho baseada em componentes revelou diferenças significativas de experimentos baseados em testes e medições de carga.
Palavras chave: confiabilidade do software; engenharia de desempenho de software; middleware orientado a mensagem; modelagem e simulação de software; Modelo de componente Palladio; desempenho do software.

I. INTRODUCCIÓN

Los métodos de predicción de rendimiento basados en modelos permiten a los arquitectos de software evaluar el desempeño de los sistemas de software durante las primeras etapas de desarrollo [1]. Tales modelos se centran en los aspectos relevantes de la arquitectura y de la lógica del negocio, dejando de lado detalles de la infraestructura subyacente. Sin embargo, estos detalles son esenciales para generar predicciones de rendimiento precisas.

La capacidad de predecir las propiedades de un artefacto con base en su diseño, sin necesidad de construirlo, es una de las características centrales de una disciplina de ingeniería [2]. En contraste con esa visión, la ingeniería del software apenas califica como una disciplina de ingeniería [3-4]: los ingenieros de software suelen carecer del entendimiento sobre el impacto de sus decisiones de diseño en los atributos de calidad, como rendimiento o confiabilidad. Como resultado, se intenta determinar la calidad del software mediante onerosos ciclos de prueba y error.

No entender el impacto de las decisiones de diseño puede ser costoso y riesgoso. En sistemas de software, un bajo rendimiento refleja una arquitectura inadecuada y no deficiencias atribuibles al código. Probar software requiere un esfuerzo previo de construcción. Si las pruebas revelan problemas de rendimiento, es muy probable que la arquitectura deba ser modificada, lo que conlleva costos adicionales.

La ingeniería de rendimiento de software (SPE, por sus siglas en inglés para *Software Performance Engineering*) es una disciplina que se centra en incorporar aspectos de rendimiento dentro del proceso de desarrollo de software, con el objetivo de entregar software confiable y con propiedades de rendimiento deseadas. Los modelos de rendimiento predictivos son una de las herramientas empleadas en SPE. Construidos en las fases tempranas del proceso de desarrollo de software, los modelos ayudan a predecir el rendimiento del software y guiar el desarrollo; para ello, los modelos de predicción de rendimiento deben considerar todos los componentes relevantes del sistema.

Para aplicaciones de software modernas, esto conlleva modelar complejas capas como máquinas virtuales o *middleware* de mensajería. Integrar todos estos modelos

puede ser costoso e ineficiente. En su lugar, es posible construir primero un modelo abstracto de la aplicación, para luego ir agregando los modelos de los componentes del sistema.

Este trabajo propone la construcción de un modelo de rendimiento para un sistema que utilice *middleware* orientado a mensajes, en busca de evaluar la influencia de éste sobre el sistema. Planteamos evaluar tal influencia mediante un caso de estudio: una aplicación de referencia cuyas métricas de rendimiento pueden ser obtenidas, que luego se adapta para utilizar *middleware* orientado a mensajes, y, entonces, se mide su rendimiento y se genera un modelo a partir de esto.

Este artículo está organizado de la siguiente manera. El resto de la Sección I presenta conceptos fundamentales relacionados con la ingeniería de rendimiento de software, el *Palladio Component Model (PCM)* y *middleware* orientado a mensajes, así como trabajos relacionados con el estudio del rendimiento de MOM utilizando modelado y simulación. *CloudStore*, la aplicación que tomamos como referencia para evaluar la influencia de MOM, se describe en la Sección II. Los resultados de la modificación del modelo PCM y su aplicación se reportan en la Sección III. El artículo concluye en la Sección IV.

A. Ingeniería de Rendimiento de Software

La caracterización de Woodside et al. [1] es comúnmente usada: “ingeniería de rendimiento del software representa toda la colección de actividades de ingeniería del software y análisis relacionados que se utilizan en el ciclo de desarrollo de software con miras a cumplir con los requerimientos de rendimiento”; según dichos autores, los enfoques para ingeniería de rendimiento pueden ser divididos en: basados en mediciones y basados en modelos.

El primero es el más común y utiliza pruebas, diagnóstico y ajustes, una vez que existe un sistema en ejecución que se pueda medir; por ello, solamente puede ser utilizado hacia el final del ciclo de desarrollo de software. Las pruebas de rendimiento habilitan la medición y usualmente se aplican después de las pruebas funcionales. Las pruebas de carga comprueban el funcionamiento de un sistema sometido a cargas de trabajo pesadas, mientras que las pruebas de rendimiento

son usadas para obtener datos cuantitativos de características de rendimiento, como tiempos de respuesta, *throughput* y utilización de recursos por un sistema en escenarios de trabajo definidos.

En contraste, el enfoque basado en modelos se centra en las etapas iniciales del desarrollo; aquí, los modelos son clave para hacer predicciones cuantitativas sobre cómo una arquitectura puede cumplir con expectativas de rendimiento (especificadas previamente). La creciente complejidad del software moderno hace difícil abordar los problemas de rendimiento en el nivel de código. Pueden requerirse cambios considerables en el diseño para resolver post-facto los problemas de rendimiento. El uso de modelos está motivado por la reducción de tales riesgos [3]. Con el modelaje se busca encontrar temprano problemas de rendimiento y alternativas de diseño, para evitar el costo y la complejidad de un rediseño. Las herramientas de modelado de rendimiento ayudan a predecir la conducta del sistema antes de que éste sea construido, así como evaluar el impacto de cambios propuestos antes de implementarlos. Se han propuesto otras clasificaciones de enfoques para SPE relacionados con la evaluación de sistemas basados en componentes [5].

En SPE, la creación y evaluación de modelos (modelado de rendimiento) es un concepto clave para evaluar cuantitativamente el rendimiento del diseño de un sistema y predecir el de otras alternativas de diseño. Tales modelos capturan aspectos relevantes del comportamiento de un sistema para identificar el efecto de cambios en la configuración o en la carga de trabajo sobre el rendimiento. Los modelos permiten predecir los efectos de los cambios sin necesidad de implementación y ejecución en un ambiente de producción, cuando el hardware con el que se cuenta sea insuficiente para soportar las cargas de trabajo [6]. En este contexto, el *Palladio Component Model* (PCM) es un enfoque de modelaje para arquitecturas de software basadas en componentes, orientado a la predicción del rendimiento. PCM contribuye al proceso de ingeniería basada en componentes y proporciona conceptos de modelaje para describir componentes de software, arquitecturas de software, instalación o despliegue (*deployment*) de componentes

y, por último, perfiles de uso de sistemas de software basados en componentes, mediante diversos submodelos [7] presentados a continuación.

- ***Especificaciones de componentes*** son descripciones abstractas y paramétricas de los componentes de software. En ellas se proporciona una descripción del comportamiento interno del componente, así como las demandas de sus recursos en RDSEFFs (*Resource Demanding Service EFFect specifications*), mediante una sintaxis similar a los diagramas de actividad de UML.
- ***Un modelo de ensamblaje*** (*assembly model*) especifica qué tipo de componentes se utilizan en una instancia de aplicación modelada, y si las instancias del componente se replican. Además, define cómo las instancias del componente se conectan para representar la arquitectura de software.
- El entorno de ejecución y los recursos, así como el despliegue de instancias de componentes para dichos contenedores de recursos, se definen en un ***modelo de asignación*** (*allocation model*).
- El ***modelo del uso*** especifica la interacción de los usuarios con el sistema de manera semejante a un diagrama de actividades de UML: una descripción abstracta de la secuencia y la frecuencia con que los usuarios activan las operaciones de un sistema.

Un modelo PCM abstrae un sistema de software en el nivel de arquitectura y es anotado con consumos de recursos que fueron medidos previamente o que han sido estimados. El modelo de componentes puede ser transformado en un modelo de análisis en particular, a fin de resolverlo analíticamente o simularlo para obtener resultados sobre el rendimiento y predicciones del sistema modelado. Los resultados del rendimiento y las predicciones se pueden utilizar como retroalimentación para evaluar y mejorar el diseño inicial, para realizar la evaluación de calidad de los sistemas de software con base en modelos [6].

B. Middleware Orientado a Mensajes

Los sistemas de mensajería que se utilizan en ambientes de negocios son denominados sistemas de mensajería empresarial o *middleware* orientados a

mensajes (MOM) [8]. MOM permite a dos o más aplicaciones intercambiar datos: paquetes autocontenidos de datos de negocio y encabezados de enrutamiento de red. Los datos de negocio contenidos en un mensaje dependen de cada aplicación o dominio; usualmente se refieren a algún tipo de transacción. La tasa de mensajes depende de la capacidad del MOM o del mediador (*broker*). El retraso depende de la latencia en el *broker* y de los caminos de entrada y salida [9].

Al usar MOM, los mensajes son trasmitidos desde una aplicación a otra a través de la red. Productos de *middleware* empresarial aseguran que los mensajes se distribuyan correctamente entre las aplicaciones. Además, estos productos proporcionan tolerancia a fallos, balanceo de carga, escalabilidad y soporte transaccional para sistemas que necesitan intercambiar de manera confiable grandes cantidades de mensajes.

Las arquitecturas de MOM son variadas: van desde las arquitecturas centralizadas que dependen de un servidor de mensajes para realizar enrutamiento, hasta arquitecturas descentralizadas que distribuyen el procesamiento del servidor hacia los clientes. Diversos protocolos, como TCP/IP, HTTP, SSL e IP, son empleados como transporte de red.

C. Trabajos relacionados

Happe et al. [7] extienden el *Palladio Component Model* para modelar *middleware* orientado a mensajes. *Performance completions* [10] proporcionan el concepto general de incluir en modelos de rendimiento los detalles de bajo nivel de ambientes de ejecución. Con la extensión del modelo, los arquitectos de software pueden especificar y configurar comunicación basada en mensajes mediante un lenguaje basado en patrones de mensajería.

El *middleware* es un factor determinante del rendimiento en sistemas distribuidos y se enfoca en su modelado y evaluación, por lo cual se propuso un enfoque basado en medición, combinado con modelos matemáticos, para predecir el rendimiento de aplicaciones J2EE. Las mediciones proporcionan los datos necesarios para calcular los valores de entrada de un modelo de red de colas. La red de colas se resuelve

para derivar métricas de rendimiento tales como tiempos de respuesta y *throughput* de la aplicación [11].

Así mismo, previamente se construyen modelos de rendimiento para *middleware* que implementa el estándar *Java Messaging Service* (JMS). Utilizan análisis de código y mediciones experimentales de implementaciones de JMS para mostrar situaciones en las que el rendimiento observado no es predicho de forma precisa por otros modelos. Finalmente, diseñan un modelo de rendimiento que captura estos efectos y se valida el modelo utilizando mediciones experimentales [12].

Por otra parte, se presenta un modelo analítico M/M/1, con políticas *first in-first out* y prioridad de colas, para evaluar el rendimiento de middleware orientado a mensajes y llamados a procedimientos remotos (RPC). Los modelos comparan el rendimiento de *middleware* orientado a mensajes y RPC con prioridad de colas y analizan su *throughput*. Los resultados demuestran que, al utilizar *middleware* orientado a mensajes con prioridad de cola, el *throughput* del sistema puede ser mejorado [13].

II. METODOLOGÍA

En esta sección se describe la aplicación *CloudStore*, que fue tomada como referencia para evaluar la aplicabilidad del modelaje basado en componentes.

A. CloudStore

CloudScale, derivado de la experiencia de PCM, es un método que permite identificar y resolver gradualmente problemas de escalabilidad en aplicaciones existentes [14-15]. El proyecto ofrece información general acerca del método, así como un entorno de desarrollo integrado creado para modelaje y análisis *(CloudScale Environment)*.

Entre las aplicaciones creadas para probar el método *CloudScale* está *CloudStore*, una aplicación web de código abierto que emula una tienda de libros en línea. La aplicación es utilizada para analizar características de los sistemas en la nube como escalabilidad, capacidad, elasticidad y eficiencia. *CloudStore* fue desarrollada en Java con la biblioteca de Spring, se ejecuta en un servidor Tomcat, utiliza una base

de datos MySQL, y su repositorio está en GitHub. Para realizar pagos, la aplicación se comunica con una pasarela de pagos, un servicio web publicado en un servidor externo a *CloudStore*, que simula el tiempo de respuesta que podría tomar realizar un pago.

Una característica de *CloudStore* atractiva para nuestro estudio es la pasarela de pagos. La interacción entre *CloudStore* y la pasarela de pagos representa un escenario de integración de sistemas. En su versión original, *CloudStore* realiza llamadas a esta pasarela de pagos de manera *ad hoc*; aquí, la introducción de comunicación basada en mensajes puede ser mejor porque:

- Permite desacoplar *CloudStore* de la pasarela de pagos: la aplicación no necesita saber detalles de ella y delega la interacción a un tercero.
- La comunicación basada en mensajes puede garantizar el envío de peticiones a la pasarela de pagos.

B. CloudStore adaptado a comunicación por mensajes

A fin de adaptar *CloudStore* con comunicación basada en mensajes para realizar los pagos, se propuso:

1. Seleccionar una solución para comunicación basada en mensajes.
2. Seleccionar biblioteca(s) de código para realizar la comunicación entre *CloudStore* y la solución seleccionada en el punto anterior.
3. Implementar un consumidor de mensajes de petición de pago: una aplicación que monitorea y procesa mensajes provenientes de la solución seleccionada en el punto #1 y los entrega a la pasarela de pagos. Esta aplicación comunicará a *CloudStore* el estado del procesamiento del pago.
4. Reemplazar el componente existente dedicado a realizar el pago contra la pasarela de pagos con nuevo código que se comunicará con la solución de mensajería.

Para el punto 1 se seleccionó ActiveMQ, un servidor de mensajería de código abierto que implementa el estándar *Java Messaging Servic*e. La aplicación

consumidora/procesadora de mensajes será expuesta como un servicio e implementada en Java.

La arquitectura adaptada de *CloudStore* para utilizar comunicación basada en mensajes se muestra en la Figura 1. Ahora, en lugar de realizar una llamada *ad hoc* a la pasarela de pagos, la aplicación se comunica con una cola de mensajería para enviar y recibir pagos. Este esquema de comunicación es conocido como *request-reply* (la aplicación que inicia la petición queda a la espera del resultado).

Para realizar un pago, *CloudStore* envía un mensaje a una cola de pagos, "Payment Queue" en la Figura 1. La aplicación consumidora de mensajes monitorea la cola para tomar los mensajes y procesarlos contra la pasarela de pagos. Una vez efectuado el pago, la aplicación consumidora envía el resultado del pago a otra cola, identificada en la Figura 1 como "Payment Status Queue". *CloudStore* monitorea los mensajes provenientes de esta cola a fin de conocer el estado del pago.

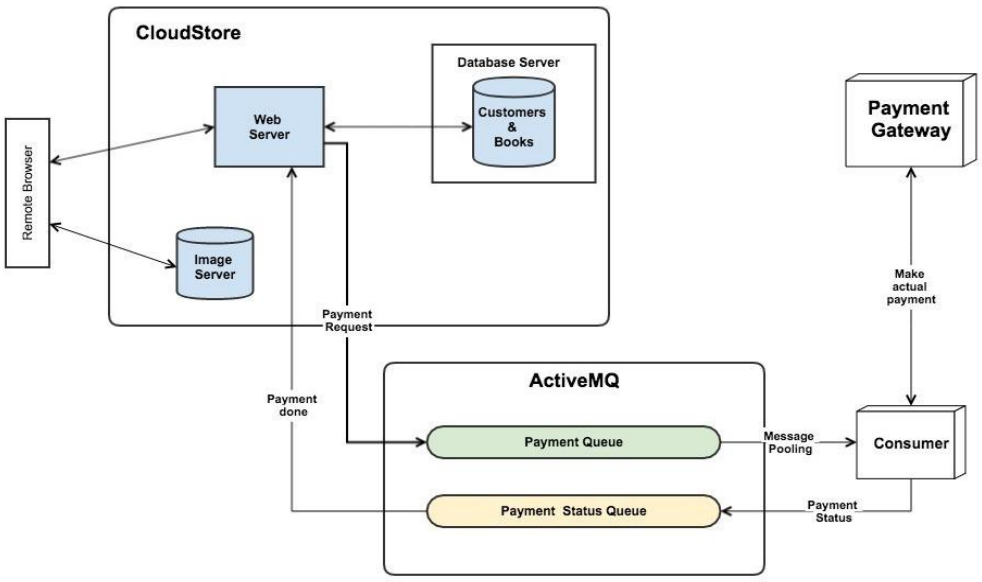

Fig. 1. Arquitectura conceptual de CloudStore utilizando comunicación basada en mensajería.

C. Modelado de MOM en CloudStore

Es necesario modificar el modelo PCM de *CloudStore* para reflejar nuevo(s) componente(s) que soporten la comunicación basada en mensajes. Para ello, se

reemplazó el componente *PaymentGateway* del modelo original con nuevos componentes para envío y recepción de mensajes. Esto con base en “plantillas” obtenidas a partir de mediciones del rendimiento de un *broker* ActiveMQ, para generar un modelo PCM, con el fin de modelar otros sistemas de mensajería [7]. Con tales plantillas se trató un MOM como una caja negra, de manera que modelos que requieran un componente MOM no necesiten los detalles de la implementación del MOM, sino proporcionar datos de interés, como el tamaño del mensaje.

En este trabajo se usó ActiveMQ para gestionar los pagos, por lo que se tomó como base esta plantilla de componentes de mensajería, los cuales se colocaron en el modelo original en sustitución del *PaymentGateway* existente. Para modelar la comunicación basada en mensajes, se usó la creación de dos componentes: un sistema de mensajes (*Message System*) y un receptor de mensajes (*Message Receiver*) [7]. Entre ambos componentes, lo que se intenta modelar es una cola FIFO. Ambos componentes fueron colocados en nuestro modelo y configuramos el tamaño del mensaje en 426Kb, después de ejecutar pruebas de carga sobre el recurso web/payment y comprobar el tamaño de mensaje enviado desde *CloudStore* hacia la pasarela de pagos. Es recomendable configurar únicamente el tamaño del mensaje por ser la característica que genera mayor diferencia en el rendimiento.

III. Resultados

En esta sección se exponen las caracterizaciones de los tiempos de respuesta asociados con los pagos, tanto sobre la aplicación *CloudStore* original como sobre la modificada para incluir *middleware* orientado a mensajes. Los resultados fueron obtenidos mediante simulaciones con base en los modelos de componentes, conforme al *Palladio* Component Model, así como pruebas de carga, para ambos sistemas de software.

A. Pruebas sobre modelo PCM y aplicación original CloudStore

A fin de caracterizar los tiempos de respuesta asociados con los pagos, usamos la simulación que trae consigo el modelo PCM de *CloudStore* para modelar la carga

sobre la página web de inicio de la aplicación, que modificamos para ejercitar la lógica de pagos. Solo fueron modificados los parámetros requeridos para dicha simulación. La simulación ejercitó el modelo con 10.000 mediciones, las cuales representarían 10.000 solicitudes de pagos. De acuerdo con los resultados de la distribución acumulada de los tiempos de respuesta esperados para las invocaciones a la lógica de pagos entregados por PCM, el 90% de las operaciones tomaron menos de 23 segundos en la simulación.

Para obtener datos sobre el comportamiento de *CloudStore* en AWS, se hicieron pruebas con JMeter para realizar 200 solicitudes durante un minuto al recurso web /payment en la aplicación original (*cs-PLAIN*), que ejercita directamente la lógica de pagos y la comunicación con la pasarela de pagos. Según los resultados obtenidos vía las pruebas de carga, el tiempo promedio de respuesta fue de 27 segundos. En la Figura 2 se muestran los tiempos de respuesta obtenidos durante la prueba.

De acuerdo con lo obtenido durante la simulación y en las pruebas de carga, el modelo PCM de *CloudStore* puede predecir el tiempo de respuesta de la lógica de pagos en un 95%. Esto sugiere que el modelo PCM proporcionado de *CloudStore* puede explicar el comportamiento de la aplicación.

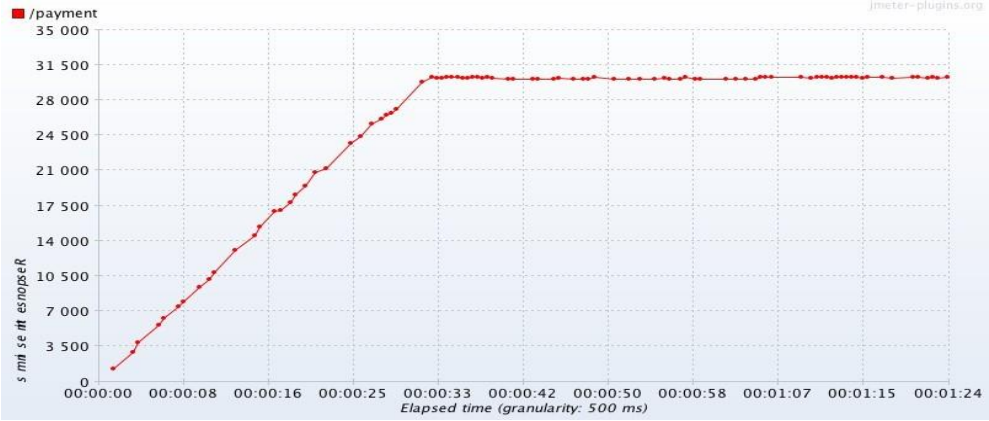

Fig. 2. Tiempos de respuesta del recurso web/payment en cs-PLAIN.

B. Pruebas sobre modelo PCM y aplicación adaptada para MOM

Con base en el modelo PCM modificado de *CloudStore* descrito anteriormente, y utilizando las mismas pruebas que trae consigo la distribución de *CloudStore*, se

procedió a ejecutar las simulaciones para explorar la influencia de los nuevos componentes de MOM en el modelo. En la Figura 3 se muestra la función de distribución acumulada para los tiempos de respuesta esperados para las invocaciones de pagos.

La prueba en JMeter que se realizó en la sección anterior fue utilizada para obtener los tiempos de respuesta de las invocaciones al recurso web/payment sobre la aplicación adaptada con MOM/ActiveMQ (*cs-JMS*). La Figura 4 muestra los tiempos de respuesta obtenidos. De acuerdo con los resultados obtenidos por las pruebas de carga, el tiempo promedio de respuesta fue de 43 segundos.

Los resultados de las simulaciones en el modelo no logran explicar el comportamiento de la aplicación. Mientras que en las pruebas de carga el tiempo promedio de una invocación al recurso web/payment tomó alrededor de 43 segundos, las simulaciones señalan que el 100% de las invocaciones no deberían de tomar más de 29 segundos. Se modificaron otros parámetros en el modelo con el fin de explorar otros resultados, pero, al ejecutar las simulaciones se obtuvieron resultados similares a los mostrados en la Figura 3.

Como observación adicional, se pudo constatar que, aunque el tiempo promedio obtenido en las invocaciones al recurso web/payment fue sustancialmente mayor en la aplicación adaptada con MOM que en la aplicación original, el 100% de las invocaciones fueron resueltas exitosamente (200 - OK). Por otro lado, en la aplicación original se obtuvieron varias respuestas infructuosas (tipo 503 - Service Unavailable) cuando se efectuaba la comunicación con la pasarela de pagos.

Dicha pasarela de pagos es una aplicación web secuencial escrita en Python, que experimentó fallos al estar expuesta a llamados de forma concurrente. En la versión adaptada con MOM/ActiveMQ, las colas de mensajería ayudaron a imponer un orden en las invocaciones a la pasarela de pagos y esto hizo que se pudieran procesar todas las invocaciones sin llegar a reportar errores. Es decir, se tiene evidencia de una mejora en la confiabilidad y disponibilidad al integrar dos sistemas vía mensajería.

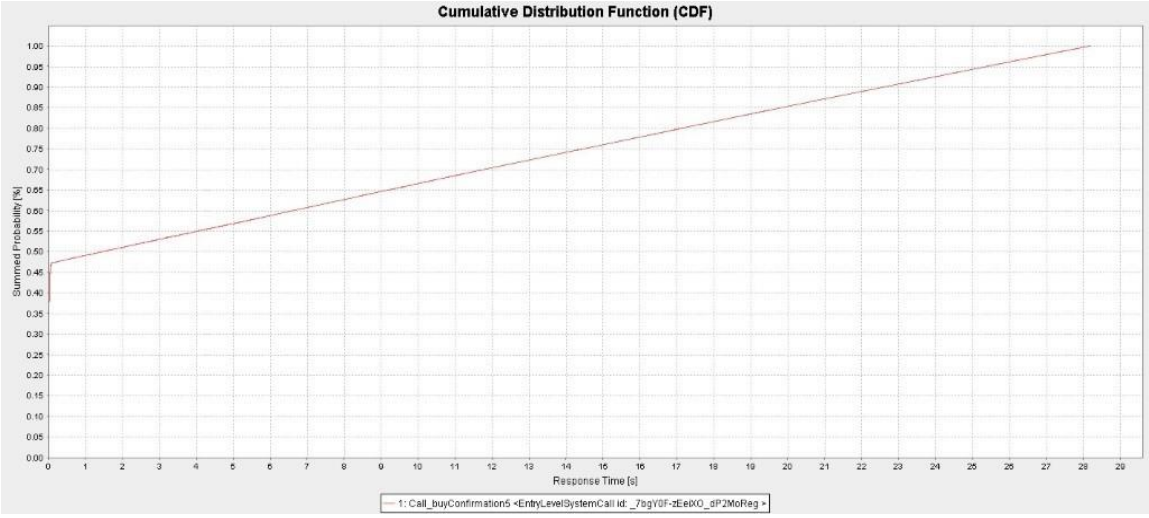

Fig. 3. Distribución acumulada de tiempos de respuesta de lógica de pagos en versión con MOM.

Fig. 4. Tiempos de respuesta del recurso web/payment en cs-JMS.

IV. CONCLUSIONES

Este artículo presenta un enfoque exploratorio para modelar y simular, con base en componentes, *middleware* orientado0020a mensajes, y determinar la influencia de éstos en el rendimiento de un sistema de software de aplicación. Se utilizó el *Palladio Component Model*, un enfoque de modelado y simulación de arquitecturas de software con base en componentes.

Aunque el rendimiento de MOM se ha estudiado previamente y varios modelos han sido propuestos, tal sistema de mediación (*middleware*) ha sido tratado como un ente aislado y no como parte de un producto de software que utiliza los servicios del

sistema de mensajería. Con el propósito de evaluar la influencia de un sistema de mensajería sobre un sistema de software completo, se adaptó una aplicación para que usara MOM, con miras a observar cambios en los tiempos de respuesta durante la ejecución. El sistema de software de base para la modificación fue *CloudStore*, una aplicación web que emula una tienda de libros en línea, que fue desarrollada como parte del proyecto *CloudScale*. Una de las ventajas de las aplicaciones que utilizan *CloudScale* es que deben contar con un modelo de componentes basado en el *Palladio Component Model* (PCM) para su procesamiento; esto posibilitó desarrollar el estudio exploratorio con base en un modelo PCM, estable y probado, como punto de partida.

Dos versiones de *CloudStore* fueron implementadas en AWS: una con el código original y otra modificada para realizar comunicación basada en mensajes con una pasarela de pagos. El modelo PCM original de *CloudStore* fue modificado con el fin de agregarle componentes para describir comunicación basada en mensajes conforme con los modelos propuestos [7]. Se ejecutaron simulaciones sobre los modelos PCM y pruebas de carga en las dos versiones de *CloudStore*. Los resultados de las simulaciones en el modelo original de *CloudStore* pudieron predecir en un 95% el comportamiento de las invocaciones a la pasarela de pagos presente en la aplicación original, mientras que observamos que los resultados de las simulaciones en el modelo adaptado con componentes para MOM, no lograron predecir el comportamiento de la aplicación que realiza la comunicación con la pasarela de pagos por medio de mensajería.

¿Por qué al introducir cambios para incorporar comunicación vía mensajería en *CloudStore* no fue posible que la simulación mediante el modelo PCM de *CloudStore* con MOM predijera los tiempos de respuesta de manera cercana a los observados en las pruebas de carga? Investigaciones posteriores a las reportadas en este trabajo revelaron la necesidad de recoger, en forma de bitácoras, datos de ejecución a partir de cargas de trabajo [16], con el fin de identificar los componentes relevantes con base en los cuales se formularía un modelo. Esto obliga a instrumentar partes del sistema, de manera que se recojan datos base de su comportamiento en cuanto a rendimiento. Estos datos pueden ser registrados en

bitácoras u otras bases, y su análisis permite extraer una versión inicial del modelo de rendimiento que puede ser posteriormente calibrado. Éste es un proceso iterativo que puede ser refinado hasta obtener estimaciones fiables de rendimiento [16].

Consideramos que *Palladio Component Model* (PCM) es un enfoque maduro, que ha sido utilizado con éxito para modelar el rendimiento de muy diversos sistemas informáticos. Para el modelado del rendimiento favorecido por el PCM, es clave identificar los componentes que realmente constituyen el sistema que se está modelando. En el caso de *CloudStore* modificado para incluir mensajería (MOM), concluimos que fueron omitidos algunos componentes que influyen en los tiempos de respuesta del sistema. Hay actividades delegadas en el MOM para interfazar la pasarela de pagos que no fueron fielmente identificadas como componentes, aunque provocan latencia en partes de ese procesamiento. En ese sentido, es necesario afinar la identificación de componentes; una vía para ello es proceder según el esquema propuesto por Flores [16], que da un marco sistemático de experimentación y modelado que va más allá del estudio exploratorio que motivó el trabajo reportado aquí.

Por otro lado, un hallazgo de esta investigación fue observar que la introducción de un mediador basado en mensajería, con un esquema de comunicación r*equest-reply* para integrarse con la pasarela de pagos, mejoró la confiabilidad de *CloudStore*, en comparación con la del sistema original. El *Palladio Component Model* también puede ser utilizado con el propósito de modelar y predecir las características de confiabilidad de arquitecturas de software en etapas tempranas de un proceso de desarrollo de software basado en componentes [3]. Modelar precisamente la confiabilidad de un sistema como *CloudStore*, con mediación del procesamiento de pagos vía mensajería, estaba fuera de los alcances de nuestro estudio exploratorio, pero es tema para una posible investigación futura.

CONTRIBUCIÓN DE LOS AUTORES

Martín Flores-González: conceptualización, análisis formal, investigación, metodología, software, visualización, redacción –borrador original, redacción–revisión y edición.

Ignacio Trejos-Zelaya: conceptualización, análisis formal, consecución de financiamiento, metodología, recursos, redacción – borrador original, redacción – revisión y edición.

César Garita: conceptualización, supervisión, redacción – borrador original, redacción – revisión y edición.

FINANCIAMIENTO

Este trabajo fue apoyado parcialmente por la Escuela de Ingeniería en Computación del Instituto Tecnológico de Costa Rica y la Fundación Tecnológica de Costa Rica.

REFERENCIAS

[1] M. Woodside, G. Franks, and D. C. Petriu. “The Future of Software Performance Engineering,” in *Future of Software Engineering*, 2007, pp. 171-187. https://doi.org/10.1109/FOSE.2007.32

[2] I. Sommerville, *Software Engineering,* London: Pearson Education, 2016.

[3] R. H. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, H. Koziolek, M. Kramer, and K. Krogmann. *Modeling and Simulating Software Architectures: The Palladio Approach,* Boston: The MIT Pressm 2016.

[4] S. Tockey, *How to Engineer Software - A Model-Based Approach*, New York: IEEE Computer Society & John Wiley & Sons, 2019.

[5] H. Koziolek, “Performance evaluation of component-based software systems: A survey,” *Performance Evaluation*, vol. 67 (8), pp. 634-658, 2010. https://doi.org/10.1016/j.peva.2009.07.007

[6] Q. Noorshams, “Modeling and Prediction of I/O Performance in Virtualized Environments,” Doctoral Thesis, Karlsruhe Institute of Technology, Karlsruhe, 2015. http://digbib.ubka.uni-karlsruhe.de/volltexte/1000046750

[7] J. Happe, H. Friedrich, S. Becker, and R. H. Reussner. “A pattern-based performance completion for Message-oriented Middleware,” in *Proceedings of the 7th international workshop on Software and performance (WOSP ’08)*, 2008, pp. 165-176. https://doi.org/10.1145/1383559.1383581

[8] M. Richards, R. Monson-Haefel, and D. Chappell, *Java Message Service*. Sebastopol, USA: O’Reilly Media, 2009.

[9] Z. B. Chew, “Modelling Message-oriented-middleware Brokers Using Autoregressive Models for Bottleneck Prediction,” Doctoral Thesis, University of London, London, 2013. http://qmro.qmul.ac.uk/xmlui/handle/123456789/8832

[10] M Woodside, D. Petriu, and K. Siddiqui. “Performance related Completions for Software Specifications,” in *Proceedings of the 24th International Conference on Software Engineering*, 2002, pp. 22-32.

[11] Y. Liu, and I. Gorton. “Performance prediction of J2EE applications using messaging protocols,” in *Proceedings of the 8th international conference on Component-Based Software Engineering (CBSE’05)*, 2005, pp. 1-16. https://doi.org/10.1007/11424529_1

[12] T., Martinec, L., Marek, A. Steinhauser, P. Tuma, Q. Noorshams, A. Rentschler, and R. Reussner. “Constructing performance model of JMS middleware platform,” in *Proceedings of the 5th ACM/SPEC international conference on Performance engineering (ICPE ’14)*, 2014, pp. 123-134. https://doi.org/10.1145/2568088.2568096

[13] S. S. Alwakeel, and H. M. Almansour, “Modeling and Performance Evaluation of Message-oriented Middleware with Priority Queuing,” *Information Technology Journal*, vol. 10, pp. 61-70, 2011. https://doi.org/10.3923/itj.2011.61.70

[14] S. Lehrig, R. Sanders, G. Brataas, M. Cecowski, S. Ivanšek, and J. Polutnik. “CloudStore — towards scalability, elasticity, and efficiency benchmarking and analysis in Cloud computing,” *Future Generation Computer Systems*, vol. 38, pp. 115-126, 2018. https://doi.org/10.1016/j.future.2017.04.018

[15] G. Brataas, E. Stav, S. Lehrig, S. Becker, G. Kopcak, and D. Huljenic. 2013. “CloudScale: scalability management for cloud systems,” in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE ’13)*, 2013, pp. 335-338. https://doi.org/10.1145/2479871.2479920

[16] M. Flores González, “Modelado y simulación de funciones en la nube en plataformas Function-as-a-Service,” Master Thesis, Instituto Tecnológico de Costa Rica, Costa Rica, 2019.