

Modelado y Simulación de Colas de Mensajes Distribuidas (*Message Oriented Middleware*)

Carlos Martín Flores González

Carné: 2015183528

Profesor: César Garita

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Maestría en Computación
MC-7205 Tema Selecto de Investigación

15 de Marzo del 2018

Índice general

Introducción	1
1. Definición del problema	3
2. Justificación	4
3. Marco Teórico	6
3.1. Revisión de la Literatura	6
3.1.1. Estrategia	6
3.1.2. Preguntas de Investigación	6
3.1.3. Estrategia de Búsqueda	6
3.1.4. Criterio de Selección	7
3.1.5. Resultado de la Revisión	7
3.2. Ingeniería de rendimiento de software (<i>Software Performance Engineering</i>)	8
3.2.1. Modelado de Rendimiento	10
3.2.2. Enfoques de ingeniería de rendimiento para software basado en componentes propuestos	10
3.2.3. Enfoques de ingeniería de rendimiento para <i>middleware</i> orienta- do a mensajes	13
3.2.4. Retos de investigación asociados a Ingeniería de rendimiento . .	13
3.2.5. Herramientas para ingeniería de rendimiento	15
3.2.6. Predicción de rendimiento dirigida por modelos	16
3.3. <i>Middleware</i> Orientado a Mensajes	17
3.3.1. Modelos de mensajería	19
3.4. Modelado de Arquitecturas de Software con el <i>Palladio Component Mo- del (PCM)</i>	20
4. Objetivos	22

4.1. Objetivo General	22
4.2. Objetivos Específicos	22
5. Entregables	23
5.1. Revisión de literatura	23
5.2. Aplicación adaptada para que soporte comunicación basada en mensajes	23
5.3. Modelo de rendimiento del nuevo sistema	24
6. Cronograma	25
Bibliografía	28

Introducción

Los métodos de predicción de rendimiento basados en modelos permiten a los arquitectos de software evaluar el rendimiento de los sistemas de software durante las primeras etapas de desarrollo. Estos modelos de predicción 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 que sean precisas.

Para los ingenieros, es una práctica común simular el modelo de un artefacto antes de construirlo. Modelos de diseños de autos, circuitos electrónicos, puentes, entre otros, son simulados para entender el impacto de decisiones de diseño en varias atributos de calidad de interés como seguridad, consumo de energía o estabilidad. La habilidad de predecir las propiedades de un artefacto en base a su diseño sin necesidad de construirlo, es una de las características centrales de una disciplina de ingeniería. A partir de esta visión de lo que se considera una disciplina de ingeniería establecida se podría decir entonces que la ingeniería de software es apenas una disciplina de ingeniería [1]. Esto porque frecuentemente los ingenieros de software carecen del entendimiento del impacto de decisiones de diseño en atributos de calidad como rendimiento o confiabilidad. Como resultado, se intenta probar la calidad del software mediante costosos ciclos de prueba y error.

El no entender el impacto en las decisiones de diseño puede ser costoso y riesgoso. Probar software significa que ya se ha hecho un esfuerzo en su implementación. Por ejemplo, si las pruebas revelan problemas de rendimiento, es muy probable que la arquitectura necesita ser modificada, lo que puede conllevar a costos adicionales. Estos costos surgen debido a que en sistemas de software empresarial un bajo rendimiento es principalmente el efecto de una arquitectura inadecuada que efecto de código.

La ingeniería de rendimiento de software (SPE por sus siglas en inglés) es una disciplina que se centra en incorporar aspecto de rendimiento dentro del proceso de desarrollo de software, con el objetivo de entregar software de confiable de acuerdo con propiedades de rendimiento particulares. 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 eventual

del software y de esta forma guiar el desarrollo, para eso los modelos de predicción de rendimiento deben capturar todos los componentes relevantes del sistema.

Para aplicaciones de software moderans, esto puede implicar modelar complejas capas tales como máquinas virtuales o *middleware* de mensajería. Componer todos estos modelos puede resultar una tarea costosa e ineficiente. En su lugar, un modelo abstracto de la aplicación se puede construir primero y luego ir agregando los modelos de los componentes del sistema.

En este trabajo se propone la construcción de un modelo de rendimiento para un sistema que utilice *middleware* orientado a mensajes con el fin de evaluar la influencia en el rendimiento de dicho sistema. Se propone evaluar esta influencia por medio de un ejemplo: tomar una aplicación de referencia con el fin de obtener sus métricas actuales de rendimiento, adaptarla para que utilice *middleware* orientado a mensajes y luego medir su rendimiento y generar un modelo a partir de esto.

Capítulo 1

Definición del problema

En los sistemas de software en los que se utiliza 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 configuración y uso para obtener predicciones significativas sobre el comportamiento de la aplicación. Sin embargo, la inclusión de un MOM en un modelo de arquitectura de software requiere un esfuerzo adicional así también como de conocimiento detallado de la infraestructura utilizada. Los arquitectos podrían llegar a omitir la influencia del MOM y esto tendría como consecuencia la generación predicciones erróneas.

Las decisiones que han sido tomadas con poca información durante las etapas de diseño usualmente son muy difíciles de alterar y podrían hacer imposible el lograr los niveles requeridos de rendimiento de un sistema una vez que este haya sido puesto en producción. Es por esto que los arquitectos y diseñadores necesitan tener la habilidad de predecir el rendimiento del MOM utilizado, trabajando a partir de diseños abstractos sin tener acceso a la implementación completa de la aplicación. Las aplicaciones que utilizan protocolos de mensajería proporcionados por un MOM realizan comunicación asíncrona y basada en colas. Estas aplicaciones deben también hacer frente a consideraciones de arquitectura adicionales como la persistencia de los mensajes y flujos de control. Todos estos factores pueden llegar a influenciar en gran medida el rendimiento de las aplicaciones.

Capítulo 2

Justificación

El *middleware* orientado a mensajes ha existido desde finales de los ochenta [2]. No solo como un estilo de comunicación entre aplicaciones sino también como un estilo de integración. De esta forma, la mensajería cumple tanto necesidades para notificaciones así como de interoperabilidad entre aplicaciones. A través de los años, los sistemas han ido creciendo significativamente en complejidad y sofisticación. La necesidad de tener sistemas con mayor confiabilidad, escalabilidad y flexibilidad que en el pasado ha dado lugar a arquitecturas más complejas. En respuesta a esta creciente demanda por sistemas más rápidos, arquitectos, diseñadores y desarrolladores han hecho uso de los sistema de mensajería como una forma de resolver estos problemas complejos.

Actualmente, el uso de MOM en arquitecturas de software modernas basadas microservicios, orientadas a eventos y de tipo *Command-Query Responsibility Segregation* (CQRS) es una solución popular que permite a las aplicaciones escalar con mayor facilidad y a la vez proporcionan integración en ambientes con tecnologías heterogéneas. Proveedores de servicios de computación en la nube como Amazon Web Services¹, Google Cloud Platform² o Microsoft Azure³ disponen de soluciones de MOM.

Uno de los principales retos del MOM es el rendimiento [3]. Muchas investigaciones se centran en el modelado de *middleware* de comunicación y su rendimiento, pero diferentes aplicaciones tienen diferentes requerimientos y esto afecta el rendimiento. Los detalles del MOM subyacente son esenciales para un entendimiento y predicción de los sistemas de software que utilizan comunicación basada en mensajes. La configuración y el uso del MOM influye fuertemente en su rendimiento y en la utilización de recursos. En arquitecturas de software modernas como las mencionadas anteriormente, el MOM es una pieza muy importante. Su operación eficiente es crucial para las comunicaciones de misión crítica. Sin embargo el MOM puede degradarse o fallar debido a una variedad de razones, lo cual lo podría llegar a convertir en un cuello de

¹Amazon Simple Queue Service(SQS) <https://aws.amazon.com/sqs/>

²Cloud Pub/Sub <https://cloud.google.com/pubsub/>

³Queue Storage <https://azure.microsoft.com/en-us/services/storage/queues/>

botella dentro del sistema en donde se encuentre [4].

El modelado de rendimiento es un enfoque útil para el análisis del rendimiento. Técnicas tradicionales de modelado de rendimiento pueden ser aplicadas a sistemas basadas en MOM [5]. Los modelos deben reflejar los efectos/comportamiento del MOM para permitir a los arquitectos de software evaluar la influencia en el rendimiento de la configuración de un MOM y de esta forma poder obtener predicciones sobre la influencia del MOM en el rendimiento del sistema.

A pesar de la importancia de contar con niveles satisfactorios de rendimiento, todavía hay una falta de enfoques de rendimiento que explícitamente tomen en cuenta las particularidades de una tecnología. Por ejemplo, en [6] se menciona que aunque el rendimiento es una necesidad inherente para lograr escalabilidad y elasticidad, ingeniería de análisis de rendimiento para microservicios ha tenido muy poca atención, y en [7] se señala que existen muchas necesidades de aplicar ingeniería de rendimiento en *DevOps*, una tendencia moderna en la que se construye y entrega software.

Con respecto a MOM, aunque se han publicado modelos de rendimiento sobre estos [8], dichos modelos se han centrado más en el MOM como componente aislado y no como un componente más dentro de un sistema, es por esto que un estudio exploratorio para determinar la influencia del MOM en el rendimiento de un sistema podría arrojar nuevo conocimiento para dar a conocer factores que favorecen o desfavorecen el uso de estas tecnologías así como representar un nuevo cuerpo de conocimiento por medio del cual se pueda evaluar la adopción e impacto del MOM durante etapas de diseño.

Capítulo 3

Marco Teórico

3.1. Revisión de la Literatura

3.1.1. Estrategia

Esta sección presenta la estrategia emprendida para cubrir el cuerpo de conocimiento relacionado con ingeniería de rendimiento de software. La estrategia general se basó en un proceso iterativo de identificación y lectura de artículos, luego identificar y leer artículos relevantes a partir de referencias y citas bibliográficas.

3.1.2. Preguntas de Investigación

Las preguntas de investigación seleccionadas para conducir la revisión de la literatura fueron:

PI1 ¿Cuáles enfoques de predicción y medición del rendimiento en sistemas de software basados en componente se han propuesto?

PI2 ¿Cuáles enfoques de predicción y medición de rendimiento de software se han utilizado para *middleware* orientado a mensajes?

PI3 ¿Qué retos y oportunidades existen con estos enfoques en la actualidad?

PI4 ¿Qué herramientas hay disponibles para modelaje de rendimiento de software?

3.1.3. Estrategia de Búsqueda

Con el fin de identificar el primer conjunto de artículos relevantes, se hizo una revisión preliminar con Google Scholar¹ porque con este motor de búsqueda se puede abarcar un amplio número de artículos y actas académicas de diferentes fuentes. El

¹<http://scholar.google.com>

criterio de búsqueda se basó en búsquedas de palabras derivadas del tema de investigación y las preguntas de investigación. Se incluyeron palabras como “component-based software”, “software performance engineering”, “software performance modeling”, “middleware oriented messages”, “modeling middleware oriented messages”, “broker modeling”, “palladio component model”, “modeling software architecture”

La búsqueda de la literatura se realizó en Marzo del 2018 usando las siguientes bases de datos electrónicas:

- *ACM Digital Library*
- *IEEE Explore Digital Library*
- Safari Books Online²
- Google Scholar

3.1.4. Criterio de Selección

Se aplicó el siguiente criterio de inclusión de artículos para esta revisión:

- Estudios que dan a conocer enfoques de análisis de rendimiento de software basado en componentes
- Estudios sobre modelaje de rendimiento de software
- Estudios sobre modelaje de rendimiento en *middleware* orientado a mensajes
- Estudios, aplicaciones o casos de uso de *Palladio Component Model*
- Estudios que reporten sobre éxito, fracaso y retos de la ingeniería de rendimiento de software basado en componentes

Los criterios de exclusión de artículos fueron los siguiente:

- Estudios relacionados ingeniería de rendimiento de software pero que no cubren o cubren muy poco aspectos sobre modelaje y simulación de arquitecturas.
- Artículos publicados hace más de 15 años atrás (rango aceptable 2003-2018).

3.1.5. Resultado de la Revisión

A continuación se presenta...

²<https://www.safaribooksonline.com>

3.2. Ingeniería de rendimiento de software (*Software Performance Engineering*)

Una definición comúnmente utilizada para definir ingeniería de rendimiento de software es la que brinda Woodside [18]: “Ingeniería de rendimiento de software (*Software Performance Engineering* - SPE) representa toda la colección de actividades de ingeniería de software y análisis relacionados utilizadas a través del ciclo de desarrollo de software que están dirigidos a cumplir con los requisitos de rendimiento”.

De acuerdo con este mismo autor, los enfoques para ingeniería de rendimiento pueden ser divididos en dos categorías: basadas en mediciones y basadas en modelos. La primera es la más común y utiliza pruebas, diagnóstico y ajustes una vez que existe un sistema en ejecución que se puede medir, es por esto que solamente puede ser utilizada conforme se va acercando el final del ciclo de desarrollo de software. Al contrario del enfoque basado en mediciones, el enfoque basado en modelos se centra en las etapas iniciales del desarrollo y iniciación con el enfoque de ingeniería de rendimiento propuesto por Smith [19]. Como el nombre lo indica, en este enfoque los modelos son clave para hacer predicciones cuantitativas de qué tan bien una arquitectura puede cumplir sus expectativas de rendimiento.

Se han propuesto otras clasificaciones de enfoques para SPE pero, con respecto a la evaluación de sistemas basados en componentes, en [14] se deja clasificación a un lado debido a que se argumenta que la mayoría de enfoques de modelaje toman alguna medición como entrada y a la mayoría de los métodos de medición los acompaña algún modelo.

Ingeniería de rendimiento basada en mediciones

Los enfoques basados en mediciones prevalecen en la industria [19] y son típicamente utilizados para verificación (¿el sistema cumple con su requisito de rendimiento?) o para localizar y arreglar *hot-spots* (cuáles son las partes que tienen peor rendimiento en el sistema). La medición de rendimiento se remota al inicio de la era de la computación, lo que ha generado una amplia gama de herramientas como generadores de carga y monitores para crear cargas de trabajo ficticias y llevar a cabo la medición de un sistema respectivamente.

Las pruebas de rendimiento aplican técnicas basadas en medición y usualmente esto es hecho luego de las pruebas funcionales o de carga. Las pruebas de carga verifican el funcionamiento de un sistema bajo 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 hardware para una configuración de un sistema bajo una carga de trabajo definida.

Ingeniería de rendimiento a través de modelado

La importancia del modelado del rendimiento está motivada por el riesgo de problemas graves de rendimiento [20] y la creciente complejidad de sistemas modernos, lo que hace difícil abordar los problemas de rendimiento al nivel de código. Cambios considerables en el diseño o en las arquitecturas pueden ser requeridos para mitigar los problemas de rendimiento. Por esta razón, la comunidad de investigación de modelado de rendimiento intenta luchar contra el enfoque de “arreglar las cosas luego” durante el proceso de desarrollo. Con la aplicación de modelo del rendimiento de software se busca encontrar problemas de rendimiento y alternativas de diseño de manera temprana en el ciclo de desarrollo, evitando así el costo y la complejidad de un rediseño o cambios en los requerimientos.

Las herramientas de modelado de rendimiento ayudan a predecir la conducta del sistema antes que este sea construido o bien, evaluar el resultado de un cambio antes de su implementación. El modelado del rendimiento puede ser usado como una herramienta de alerta temprana durante todo el ciclo de desarrollo con mayor precisión y modelos cada vez más detallados a lo largo del proceso. Al inicio del desarrollo un modelo no puede ser validado contra un sistema real, por esto el modelo representa el conocimiento incierto del diseñador. Como consecuencia de esto el modelo hace suposiciones que no necesariamente se van a dar en el sistema real, pero que van a ser útiles para obtener una abstracción del comportamiento del sistema. En estas fases iniciales, la validación se obtiene mediante el uso del modelo, y existe el riesgo de conclusiones erróneas debido a su precisión limitada. Luego, el modelo puede ser validado contra mediciones en el sistema real (o parte de este) o prototipos y esto hace que la precisión del modelo se incremente.

En [21] se sugiere que los métodos actuales tienen que superar un número de retos antes que puedan ser aplicados en sistemas existentes que enfrentan cambios en su arquitectura o requerimientos. Primero, debe quedar claro cómo se obtienen los valores para los parámetros del modelo y cómo se pueden validar los supuestos. Estimaciones basadas en la experiencia para estos parámetros no son suficientes y mediciones en el sistema existente son necesarias para hacer predicciones precisas. Segundo, la caracterización de la carga del sistema en un entorno de producción es problemática debido a los recursos compartidos (bases de datos, hardware). Tercero, deben desarrollarse métodos para capturar parámetros del modelo dependientes de la carga. Por ejemplo un incremento en el tamaño de la base de datos probablemente incrementará las necesidades de procesador, memoria y disco en el servidor.

Técnicas comunes de modelado incluyen redes de colas, extensiones de estas como redes de colas en capas y varios tipos de redes de Petri y procesos de álgebra estocástica.

3.2.1. Modelado de Rendimiento

En SPE, la creación y evaluación de modelos de rendimiento es un concepto clave para evaluar cuantitativamente el rendimiento del diseño de un sistema y predecir el rendimiento de otras alternativas de diseño. Un modelo de rendimiento captura el comportamiento relevante al rendimiento de un sistema para identificar el efecto de cambios en la configuración o en la carga de trabajo en el rendimiento. Permite predecir los efectos de tales cambios sin necesidad de implementarlo y ponerlo en producción, que podrían ser no solamente tareas costosas sino también un desperdicio en el caso que un el hardware con el que se cuenta pruebe ser insuficiente para soportar la intensidad de la carga de trabajo. [12]

La forma del modelo de rendimiento puede comprender desde funciones matemáticas a formalismos de modelado estructural y modelos de simulación. Estos modelos varían en sus características clave, por ejemplo, las suposiciones de modelado de los formalismos, el esfuerzo de modelado requerido y el nivel de abstracción.

En cuanto a técnicas de simulación, a pesar que estas permiten un estudio más detallado de los sistemas que modelos analíticos, la construcción de un modelo de simulación requiere de conocimiento detallado tanto de desarrollo de software como de estadística [19]. Los modelos de simulación también requieren usualmente de mayor tiempo de desarrollo que los modelos analíticos. En [18] se menciona que “la construcción de un modelo de simulación es caro, algunas veces comparable con el desarrollo de un sistema, y, los modelos de simulación detallados puede tardar casi tanto en ejecutarse como el sistema.

3.2.2. Enfoques de ingeniería de rendimiento para software basado en componentes propuestos

La encuesta llevada a cabo en [14] proporciona una clasificación de enfoques de medición y predicción de rendimiento para sistemas de software basados en componentes. Otra clasificación es la que se expone en [22] donde se presenta una revisión de métodos de predicción de rendimiento basado en modelos para sistemas en general, pero no analiza los requerimientos propios para sistemas basados en componentes.

De acuerdo con [14] durante los últimos diez años, los investigadores han propuesto muchos enfoques para evaluar el rendimiento (tiempos de respuesta, *throughput*, utilización de recursos) de sistemas de software basados en componentes. Estos enfoques tratan con predicción de rendimiento y medición del rendimiento. Los primeros analizan el rendimiento esperado de un diseño de software basado en componentes para evitar problemas de rendimiento en la implementación del sistema, lo que podría llevar a costos substanciales para rediseñar la arquitectura. Los otros analizan el rendimiento observable de sistemas de software basados en componentes implementados para

entender sus propiedades, determinar su capacidad máxima, identificar componentes críticos y para remover cuellos de botella.

Métodos de evaluación de rendimiento

Los enfoques se agruparon en dos grandes grupos: enfoques principales que proporcionan procesos de evaluación de rendimiento completo y enfoques suplementarios que se centran en aspectos específicos como medición de componentes individuales o modelaje de las propiedades de rendimiento de los conectores de un componente.

Enfoques principales

- **Enfoques de predicción basados en UML:** los enfoques en este grupo se enfocan en la predicción de rendimiento en tiempo de diseño para sistemas de software basado en componentes modelados con el Lenguaje de Modelado Unificado (UML por sus siglas en inglés). UML 2.0 tiene la noción de componente de software como una clase extendida. UML permite modelar el comportamiento de un un componente con diagramas de secuencia, actividad y colaboración. El alojamiento de los componentes puede ser descrito como con diagramas de despliegue(*deployment*).
 - CB-SPE - *Component-Based Software Performance Engineering*
- **Enfoques de predicción basados en Meta-Modelos propietarios:** Los enfoques en este grupo apuntan a las predicciones de rendimiento de tiempo de diseño. En lugar de usar UML como lenguaje de modelado para desarrolladores y arquitectos, estos enfoques tienen meta-modelos propietarios.
 - CBML - *Component-Based Modeling Language*
 - PECT - *The Prediction Enabled Component Technology*
 - COMQUAD - *Components with Quantitative properties and Adaptivity*
 - KLAPPER
 - ROBOCOP
 - Palladio
- **Enfoques de predicción centrados en *middleware*:** hacen énfasis en la influencia del *middleware* en el rendimiento de un sistema basado en componentes. Por lo tanto miden y modelan el rendimiento de plataformas *middleware* como JavaEE o .Net. Se basan en la suposición que la lógica de negocio de los componentes como tal tienen poco impacto en el rendimiento general del sistema y por eso no requieren un modelado detallado.

- NICTA
- **Enfoques basados en especificaciones formales:** estos enfoques siguen teorías fundamentales de especificación de rendimiento y no toman en cuenta marcos de trabajo (*frameworks*) de medición y predicción.
 - RESOLVE
 - HAMLET
- **Enfoques de monitoreo para sistemas implementados:** asumen que un sistema basado en componentes ha sido implementado y puede ser probado. El objetivo es encontrar problemas de rendimiento en un sistema en ejecución, identificar cuellos de botella y adaptar el sistema para que pueda lograr los requerimientos de rendimiento.
 - COMPAS
 - TESTEJB
 - AQUA
 - PAD

Enfoques Suplementarios

- **Enfoques de monitoreo para componentes implementados:** El objetivo de los enfoques de medición para implementaciones de componentes de software individuales es derivar especificaciones de rendimiento parametrizadas a través de mediciones múltiples. El objetivo es obtener el perfil de uso, dependencias y la plataforma de implementación a partir de la especificación de rendimiento, de modo que pueda usarse en diferentes contextos.
 - RefCAM
 - COMAERA
 - ByCounter
- **Enfoques de predicción con énfasis en conectores de componentes:** Estos enfoques asumen un lenguaje de descripción de componentes existente y se centran en modelar y medir el impacto en el rendimiento de las conexiones entre componentes. Estas conexiones pueden implementarse con diferentes técnicas *middle-ware*.
 - Verdickt
 - Grassi
 - Becker

- Happe

Recientemente varios enfoques de predicción basados en meta-modelos propietarios han sido propuestos para optimización de diseño de arquitecturas, modelado de calidad de servicio y escalabilidad. PerOpteryx [23] es un enfoque de optimización de diseño de arquitecturas que manipula modelos especificados en *Palladio Component Model* (PCM) [1] y utiliza el algoritmo evolucionario multi-objetivo NSGA-II. Para análisis de rendimiento utiliza redes de colas en capas. *Descartes Modeling Language* [24] es un lenguaje de modelado de arquitecturas para modelar calidad de servicio y aspectos relacionados con la gestión de recursos de los sistemas, las infraestructuras y los servicios de tecnología de información dinámicos modernos. *CloudScale*³ [25] es un enfoque de diseño y evolución de aplicaciones y servicios escalables en la nube. En CloudScale se identifica y gradualmente se resuelven problemas de escalabilidad en aplicaciones existentes y también permite el modelado de alternativas de diseño y el análisis del efecto de la escalabilidad en el costo. Cabe mencionar que estos últimos enfoques han sido influenciado en gran medida por el trabajo llevado a cabo en PCM.

3.2.3. Enfoques de ingeniería de rendimiento para *middleware* orientado a mensajes

3.2.4. Retos de investigación asociados a Ingeniería de rendimiento

Informes recientes como [6] y [7] señalan que los principales retos en investigación de ingeniería de rendimiento en la actualidad están asociados con generación de modelos a partir de código, integración de la evaluación de rendimiento en el ciclo de desarrollo y modelado de rendimiento de sistemas en la nube.

Con respecto a integración en ciclos de desarrollo de software y principalmente en aquellos basados en DevOps, una tendencia hacia una estrecha integración entre equipos desarrollo y operaciones, en [7] se destaca que la necesidad de tal integración está orientada por el requerimiento de adaptar aplicaciones empresariales a los cambios del ambiente del negocio continuamente. El rendimiento describe las propiedades del sistema con respecto a su ejecución y uso de recursos. Métricas comunes son tiempo de respuesta, *throughput* y la utilización de los recursos. Los objetivos de rendimiento para aplicaciones empresariales son típicamente definidos al establecer cotas superiores/inferiores para estas métricas y transacciones de negocio específicas.

Actividades de administración del rendimiento

Evaluación del rendimiento basado en modelos:

³<http://www.cloudscale-project.eu/>

- La representación de la memoria principal y del recolector de basura aún no están explícitamente integrados ni considerados en los modelos de rendimiento
- La selección de técnicas apropiadas de solución requiere de mucha experiencia

Extracción de modelos de rendimiento y de cargas de trabajo:

- La precisión de los modelos puede llegar a expirar si no son actualizados cuando hay cambios. Mecanismos de detección son requeridos para aprender cuando los modelos son antiguos y hay que actualizarlos.
- La extracción de capacidades de rendimiento se basa en una combinación de software y los recursos de hardware en los que se implementa. Este enfoque combinado es compatible con la precisión de la predicción, pero está menos calificado con respecto a la portabilidad de los conocimientos a otras plataformas. Una dirección de investigación futura podría ser extraer modelos separados (por ejemplo, *middleware* y modelos de aplicación separados).

Modelos de rendimiento durante etapas diseño

- Los retos de utilizar modelos de rendimiento en etapas tempranas de desarrollo es que usualmente es difícil validar la precisión de los modelos hasta que un sistema en ejecución exista. Las predicciones de rendimiento basadas en suposiciones, entrevistas y pruebas previas pueden ser también imprecisas y por tanto las decisiones que se hagan a partir de estas predicciones.

En relación con sistemas y arquitecturas en la nube como las basadas en microservicios otras tecnologías importantes para *deployment* como, virtualización basada en contenedores y soluciones de orquestación de contenedores han emergido. Estas tecnologías permiten explotar plataformas en la nube, proporcionando altos niveles de escalabilidad, disponibilidad y portabilidad para microservicios. A pesar del hecho de que es una necesidad inherente contar con escalabilidad y elasticidad, la ingeniería de rendimiento para microservicios hasta ahora ha tenido poca atención por parte de las comunidades tanto de microservicios como de comunidades de investigación de ingeniería de rendimiento [6]. Un gran cuerpo de conocimiento y buenas prácticas para ingeniería de rendimiento para desarrollo tradicional de software y arquitecturas está disponible. Sin embargo, sus aplicaciones en DevOps imponen tanto retos como oportunidades.

Retos de investigación en microservicios

Pruebas de rendimiento

- Reemplazar y compensar pruebas extensivas de integración y de sistema por un control detallado de los entornos de producción
- Alinear las pruebas de rendimiento y pruebas de regresión de rendimiento con prácticas de entrega continua, por ejemplo, acelerar las estas de pruebas correspondientes.
- Selección dinámica y semi automática de pruebas de rendimiento

Monitoreo

- Instrumentación para monitoreo distribuido arquitecturas microservicios políglotas
- Métricas adicionales para monitorear microservicios
- Técnicas de detección precisa de anomalías en arquitecturas de microservicios

Modelado del rendimiento: de momento no existen enfoques para modelado del rendimiento de microservicios

- Adoptar modelos de rendimiento para casos de uso como planeamiento de capacidad, confiabilidad y resiliencia
- Buscar abstracciones de modelado apropiadas
- Extracción automática de modelos de rendimiento
- Aprender del comportamiento de la infraestructura e integrarlo en los modelos de rendimiento

3.2.5. Herramientas para ingeniería de rendimiento

—

Un atributo no funcional que es usualmente es de alta importancia durante el desarrollo de software es el rendimiento del sistema en cuestión. Si los sistemas sufren de un rendimiento insuficiente, esto hace que usualmente no se utilicen como se esperaba o bien hacen que los proyectos fallen. Por lo tanto, la predicción del rendimiento de los sistemas de software está en el centro de agendas de investigación desde finales de los años noventa [10]. El término rendimiento a menudo caracteriza el comportamiento en el tiempo y la eficiencia de los recursos de los sistema de hardware y de software. Las propiedades de rendimiento más importantes son tiempos de respuesta, *throughput* y utilización de recursos.

La ingeniería de rendimiento de software se dedica a evaluar el rendimiento de sistemas de software al ofrecer diferentes métodos como modelado analítico, simulación

y toma de mediciones. El objetivo de SPE es conducir evaluaciones del rendimiento de arquitecturas de software tan pronto como se pueda.

En SPE, la creación y evaluación de modelos de rendimiento es un concepto clave para evaluar cuantitativamente el rendimiento del diseño de un sistema y predecir el rendimiento de otras alternativas de diseño. Un modelo de rendimiento captura el comportamiento relevante al rendimiento de un sistema para identificar el efecto de cambios en la configuración o en la carga de trabajo en el rendimiento. Permite predecir los efectos de tales cambios sin necesidad de implementarlo y ponerlo en producción, que podrían ser no solamente tareas costosas sino también un desperdicio en el caso que un el hardware con el que se cuenta pruebe ser insuficiente para soportar la intensidad de la carga de trabajo.

El punto de inicio de la mayoría de enfoques de SPE es la arquitectura del sistema de software. La arquitectura puede ser mejorada con anotaciones de rendimiento. Este modelo de software debe ser transformado en un modelo de rendimiento –como por ejemplo redes de colas, redes de Petri o cadenas de Markov – para luego ser resuelto para las métricas de interés.

En el área de SPE, la idea de usar tecnologías orientadas a modelos ha ganado atención porque estas proporcionan transformación automática desde algún origen hacia un modelo de rendimiento. Sin embargo, los métodos de predicción de rendimiento basados en modelos requieren meta-modelos adecuados. Estos meta-modelos formalizan la sintáxis y semántica del modelo origen y destino. La transformación y procesamiento automático simplifica el enfoque de ingeniería de rendimiento de software y lo hace menos propenso a errores.

3.2.6. Predicción de rendimiento dirigida por modelos

La predicción del rendimiento dirigida por modelos permite a los arquitectos de software especificar modelos de rendimiento en un lenguaje específico a su dominio. Esto puede a partir de modelos UML anotados con información relevante al rendimiento o por medio de lenguajes de descripción de arquitecturas especializados para la predicción de rendimiento como *Palladio Component Model* (PCM) [1, 11]. Para derivar métricas de rendimiento, el modelo de software se transforma en un modelo de rendimiento, como se muestra en la figura 3.2. Las métricas de rendimiento derivadas del modelo de rendimiento deberían traducirse nuevamente al modelo de diseño, para permitir una interpretación fácil para los arquitectos de software.

3.3. *Middleware* Orientado a Mensajes

Junto con el crecimiento de Internet, los sistemas distribuidos han crecido a una escala masiva. Hoy en día, estos sistemas puede involucrar a miles de entidades distribuidas globalmente. Esto ha motivado el estudio de modelos de comunicación y sistemas flexibles, que puedan reflejar la naturaleza dinámica y desacoplada de las aplicaciones.

La integración de tecnologías heterogéneas es una de las áreas principales en donde la comunicación basada en mensajes juega un papel clave. Más y más compañías se enfrentan al problema de integrar sistemas y aplicaciones heterogéneas dentro y fuera de la organización ya sea por fusiones, adquisiciones, requisitos comerciales o simplemente un cambio en la dirección tecnológica. No es raro encontrar una gran cantidad de tecnologías y plataformas dentro de una sola compañía, desde productos de código libre y comerciales hasta sistemas y equipos heredados (*legacy*).

La comunicación basada en mensajes también ofrece la habilidad de procesar solicitudes de manera asincrónica, proporcionando a los arquitectos y desarrolladores soluciones para reducir o eliminar cuellos de botella en un sistema e incrementar la productividad del usuario final y del sistema en general. Dado que la comunicación basada en mensajes provee un alto grado de desacoplamiento entre componentes, los sistemas que utilizan esta tecnología también logran contar con altos grados de agilidad y flexibilidad en su arquitectura.

A los sistemas de mensajería de aplicación-a-aplicación que se utilizan sistemas de negocios se les denomina genéricamente sistemas de mensajería empresarial o *middleware* orientado a mensajes [9]. MOM permite a dos o más aplicaciones intercambiar información en forma de mensajes. Un mensaje, en este caso, es un paquete autocontenido de datos de negocio y encabezados de enrutamiento de red. Los datos de negocio contenidos en un mensaje puede ser cualquier cosa, van a depender de cada negocio, y usualmente contiene información acerca de algún tipo de transacción. En sistemas de mensajería empresariales, los mensajes informan a una aplicación de la ocurrencia de algún evento en otro sistema. La tasa de mensajes depende de la capacidad de la implementación de MOM o *broker*. El retraso depende de la latencia en el *broker* y en los caminos de entrada y salida [4].

Al usar MOM, los mensajes son transmitidos 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 usualmente proporcionan tolerancia a fallos, balanceo de carga, escalabilidad y soporte transaccional para sistemas que necesitan que necesitan intercambiar de manera confiable grandes cantidades de mensajes.

Los fabricantes de MOM usan diferentes formatos de mensajes y protocolos de red para intercambiar mensajes pero la semántica básica es la misma. Una interfaz de pro-

gramación de aplicación (API, por sus siglas en inglés) se utiliza para crear un mensaje, cargar los datos, asignar información de enrutamiento y enviar el mensaje. La misma API se utiliza para recibir los mensajes producidos por otras aplicaciones.

En todos los sistemas modernos de mensajería empresarial, las aplicaciones intercambian mensajes a través de canales virtuales llamados destinos (*destinations*). Cuando un mensaje se envía, se dirige a un destino (por ejemplo una cola o un tópico) no a una aplicación específica. Cualquier aplicación que suscriba o registre un interés en ese destino puede recibir el mensaje. De esta forma, las aplicaciones que reciben mensajes y aquellas que envían mensajes están desacopladas. Los emisores y receptores no están enlazados uno con otro de ninguna forma y pueden enviar y recibir mensajes como mejor les parezca.

Un concepto clave de MOM es que los mensajes son entregados de forma asincrónica desde un sistema a otros sobre la red. El entregar un mensaje de manera asincrónica significa que el emisor no requiere esperar a que el mensaje sea recibido o manejado por el receptor, él es libre de enviar el mensaje y continuar su procesamiento. Los mensajes asincrónicos son tratados como unidades autónomas: cada mensaje es autocontenido y lleva consigo todos los datos necesarios para ser procesado.

En comunicación de mensajes asincrónicos, las aplicaciones usan una API para contruir un mensaje, luego pasarlos al MOM para su entrega a uno o varios recipientes (Figura 3.1). Un mensaje es un paquete de datos que es enviado desde una aplicación a otra sobre la red. El mensaje debe de ser autodescriptivo en el sentido que debe de contener todo el contexto necesario para que permit a los recipientes llevar a cabo su trabajo de forma independiente.

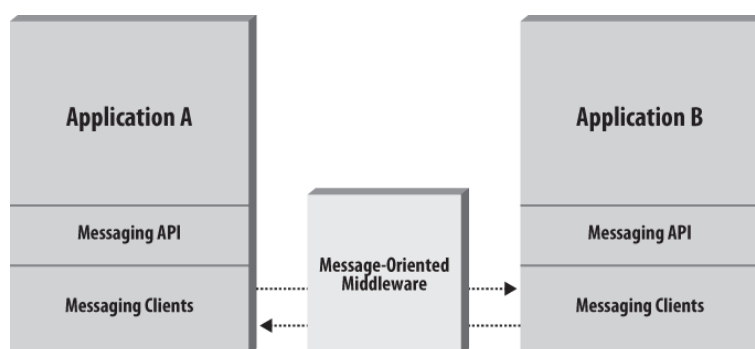


Figura 3.1: *Middleware* Orientado a Mensajes. Tomado de [9]

La arquitecturas de MOM de hoy en día varían en su implementación. Van desde las arquitecturas centralizadas que dependen de un servidor de mensajes para realizar enrutamiento a arquitecturas descentralizadas que distribuyen el procesamiento del servidor hacia los clientes. Una variedad de protocolos como TCP/IP, HTTP, SSL y IP son empleados como capa de transporte de red.

Los sistemas de mensajería están compuestos por clientes de mensajería (*messa-*

ging clients) y algún tipo de servidor MOM. Los clientes envían mensajes al servidor de mensajería el cual distribuye estos mensajes a otros clientes. El cliente es una aplicación de negocio o componente que es usa una API de mensajería.

3.3.1. Modelos de mensajería

Punto-a-Punto

Este modelo de mensajería permite a los clientes enviar y recibir mensajes de forma síncrona como asíncrona a través de canales virtuales conocidos como colas. En este modelo a los productores de mensajes se les llama emisores (*senders*) y a los consumidores se les conoce como receptores (*receivers*). El modelo punto-a-punto ha sido tradicionalmente un modelo basado en *polling*, en donde los mensajes son solicitados desde la cola en lugar de ser puestos en el cliente automáticamente. Los mensajes que se envían a una cola son recibidos por uno y solo un *receiver*, aunque pueden haber otros *receivers* escuchando en la cola por el mismo mensaje. Este es un modelo que promueve acoplamiento esto porque generalmente el *sender* conocer cómo el mensaje va a ser utilizado y quién lo va a recibir.

Publish-and-Subscribe

En este modelo, los mensajes son publicados en un canal virtual llamado tópicos. Los productores de mensajes son conocidos como *publishers* y a los consumidores se les llama *subscribers*. Los mensajes pueden ser recibidos por múltiples *subscribers*, a diferencia del modelo punto-a-punto. Cada *subscriber* recibe una copia de cada mensaje. Este es un modelo basado en *push* en donde los mensajes son automáticamente transmitidos a los consumidores sin que estos tengan que solicitarlos o revisar la cola por nuevos mensajes.

Este modelo tiende a ser menos acoplado que el modelo punto-a-punto debido a que el publicador del mensaje generalmente no está conciente de cuántos subscriptores hay y lo qué van a hacer estos con el mensaje.

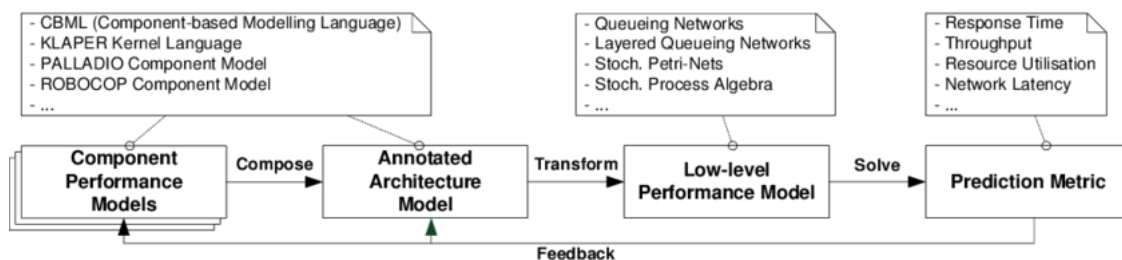


Figura 3.2: Predicción de rendimiento dirigida por modelos. Tomado de [13]

3.4. Modelado de Arquitecturas de Software con el *Palladio Component Model (PCM)*

El *Palladio Component Model* es un enfoque de modelaje para arquitecturas de software basados en componentes que permite predicción de rendimiento basada en modelos. PCM contribuye el proceso de desarrollo de ingeniería basado en componentes y proporciona conceptos de modelaje para describir componentes de software, arquitectura de software, despliegue (*deployment*) de componentes y perfiles de uso de sistemas de software basados en componentes en diferentes submodelos (Figura 3.3).

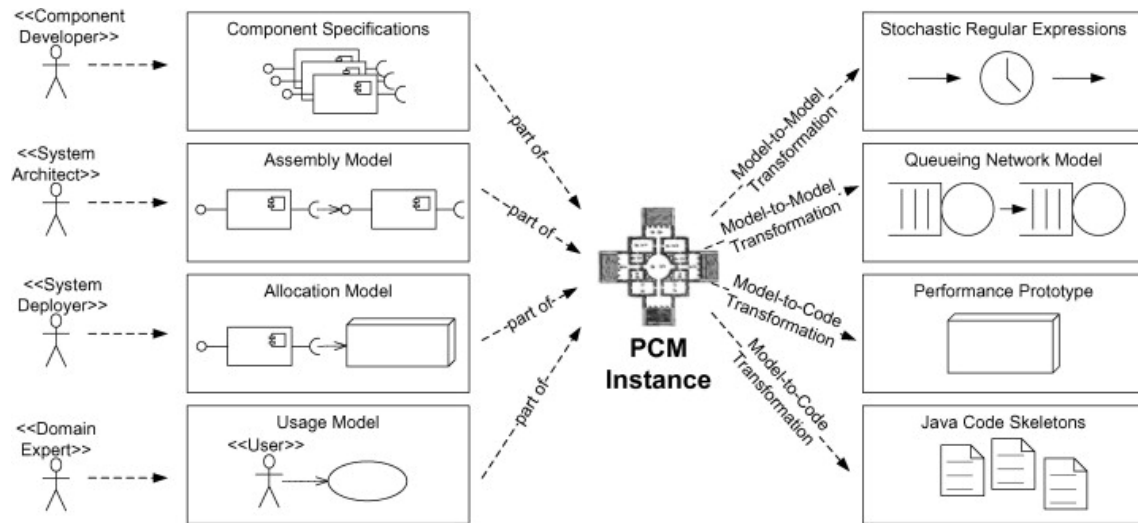


Figura 3.3: Instancia de un modelo PCM. Tomado de [13]

- **Especificaciones de componentes** son descripciones abstractas y paramétricas de los componentes de software. En las especificaciones de software se proporciona una descripción del comportamiento interno del componente así como las demandas de sus recursos en RDSEFFs (*Resource Demanding Service Effect specifications*) utilizando 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 representando la arquitectura de software.
- El entorno de ejecución y los recursos, así como el despliegue (*deployment*) 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 utilizando una sintaxis similar al diagrama de actividades de UML proporcionando una

descripción abstracta de la secuencia y la frecuencia en que los usuarios activan las operaciones disponibles en un sistema.

Un modelo PCM abstrae un sistema de software a nivel de arquitectura y se anota con consumos de recursos que fueron medidos previamente u otros que son estimados. El modelo puede entonces ser usado en transformaciones de modelo-a-modelo o modelo-a-texto a un modelo de análisis en particular (redes de colas o simulación de código) que puede ser analíticamente resuelto o simulado para obtener resultados sobre el rendimiento y predicciones del sistema modelado. Los resultados del rendimiento y las predicciones pueden ser utilizadas como retroalimentación para evaluar y mejorar el diseño inicial, permitiendo así una evaluación de calidad de los sistemas de software en base a un modelo [12].

Capítulo 4

Objetivos

4.1. Objetivo General

Diseñar un método para modelar sistemas distribuidos de intercambio de mensajes (*message-oriented middleware*) para evaluar la influencia que tienen en el rendimiento de un sistema de software.

4.2. Objetivos Específicos

1. Revisión del estado del arte de trabajos relacionados con enfoques de predicción y medición del rendimiento en sistemas de software basados en componentes y en *middleware* orientado a mensajes
2. Adaptar un sistema de software de referencia para el cual ya exista un modelo de rendimiento y simulaciones para que se integre y comunique con *middleware* orientado a mensajes.
3. Comparar la solución implementada bajo diferentes cargas de trabajo
4. Crear un modelo del nuevo sistema y su rendimiento
5. Validar y analizar el modelo creado a través de experimentos

Capítulo 5

Entregables

5.1. Revisión de literatura

Alineado con objetivo específico 1

Se pretende identificar los resultados de otros estudios relacionados con modelaje de rendimiento de software, así como retos y oportunidades de investigación que existan en esta área. Las preguntas de investigación inicialmente propuestas para conducir esta revisión son las siguientes:

PI1 ¿Cuáles enfoques de predicción y medición del rendimiento en sistemas de software basados en componente se han propuesto?

PI2 ¿Cuáles enfoques de predicción y medición de rendimiento de software se han utilizado para *middleware* orientado a mensajes?

PI3 ¿Qué retos y oportunidades existen con estos enfoques en la actualidad?

PI4 ¿Qué herramientas hay disponibles para modelaje de rendimiento de software?

5.2. Aplicación adaptada para que soporte comunicación basada en mensajes

Alineado con objetivos específicos 2 y 3

Una vez que se haya identificado un sistema de software del cual ya exista un modelo de rendimiento y simulaciones, este se tomará como base para adaptarlo de tal forma que utilice *middleware* orientado a mensajes en algún punto de su ejecución.

A la aplicación adaptada se le realizarán mediciones de su rendimiento con el fin de tener estos como entrada para las subsecuentes tareas de modelado propuestas. Serán estas mediciones las que ayudan a refinar y calibrar el modelo.

5.3. Modelo de rendimiento del nuevo sistema

Alineado con objetivos específicos 4 y 5

A partir del sistema adaptado y sus mediciones, se modificará el modelo de rendimiento original del sistema de referencia para reflejar los cambios introducidos por el *middleware* orientado a mensajes.

Este nuevo modelo se ejercitará con los perfiles de uso de la aplicación de referencia, con el fin de validar si este puede aproximar el comportamiento del sistema ahora que se le han introducido cambios.

Capítulo 6



Cronograma

Se cuentan con aproximadamente 11 semanas para llevar a cabo este trabajo. Las primeras dos semanas se van a dedicar a hacer una revisión de la literatura relevante.

Luego de esto, se dedicarán cuatro semanas a seleccionar una aplicación de referencia y adaptarla para que utilice comunicación basada en mensajes en algún punto de su ejecución. Una vez adaptada se ejecutarán pruebas de carga para determinar cuáles es su rendimiento.

La última etapa de la investigación será la de crear un modelo del rendimiento del nuevo sistema utilizando PCM. A este modelo se le ejecutarán simulaciones de uso con el fin de ir calibrando su precisión.

A continuación se presenta el calendario de actividades propuesto para esta investigación:

		Name	Duration	Start	Finish	Predecessors
1		Revisión de la Literatura	10 days	3/19/18 8:00 AM	3/30/18 5:00 PM	
2		Selección de literatura relevante	5 days	3/19/18 8:00 AM	3/23/18 5:00 PM	
3		Preparacion del documento	5 days	3/26/18 8:00 AM	3/30/18 5:00 PM	2
4		Adaptación de aplicación	21 days	4/2/18 8:00 AM	4/30/18 5:00 PM	1
5		Selección de aplicación de referencia	5 days	4/2/18 8:00 AM	4/6/18 5:00 PM	
6		Adaptar aplicación para que utilice MOM	13 days	4/9/18 8:00 AM	4/25/18 5:00 PM	5
7		Ejecutar pruebas de carga/rendimiento	3 days	4/26/18 8:00 AM	4/30/18 5:00 PM	6
8		Modelado de rendimiento	20 days	5/1/18 8:00 AM	5/28/18 5:00 PM	4
9		Modelado de aplicación con MOM	10 days	5/1/18 8:00 AM	5/14/18 5:00 PM	
10		Ejecución de pruebas de rendimiento	10 days	5/15/18 8:00 AM	5/28/18 5:00 PM	9
11		Preparación documento final	3 days	5/29/18 8:00 AM	5/31/18 5:00 PM	8

Bibliografía

- [1] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, Anne Koziolk, Heiko Koziolk, Max Kramer, and Klaus Krogmann. *Modeling and Simulating Software Architectures: The Palladio Approach*. The MIT Press. 2016.
- [2] Bruce Snyder, Dejan Bosnanac, Rob Davies. *ActiveMQ in Action*. Manning, 2011.
- [3] S.S. Alwakeel and H.M. Almansour. *Modeling and Performance Evaluation of Message-oriented Middleware with Priority Queuing*. Information Technology Journal, 10: 61-70. 2011. DOI <http://dx.doi.org/10.3923/itj.2011.61.70>
- [4] Chew, Zen Bob. *Modelling Message-oriented-middleware Brokers Using Autoregressive Models for Bottleneck Prediction*. PhD Thesis. Queen Mary, University of London. 2013. <https://qmro.qmul.ac.uk/jspui/handle/123456789/8832>
- [5] Yan Liu and Ian Gorton. *Performance prediction of J2EE applications using messaging protocols*. In Proceedings of the 8th international conference on Component-Based Software Engineering (CBSE'05), George T. Heineman, Ivica Crnkovic, Heinz W. Schmidt, Judith A. Stafford, and Clemens Szyperski (Eds.). Springer-Verlag, Berlin, Heidelberg, 1-16. 2005. DOI http://ezproxy.itcr.ac.cr:2075/10.1007/11424529_1
- [6] Robert Heinrich, André van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, and Johannes Wettinger. *Performance engineering for microservices: Research challenges and directions*. In Companion Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, 2017, pages 223-226. DOI: <https://doi.org/10.1145/3053600.3053653>
- [7] Andreas Brunnert, André van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Roman Herbst, Pooyan Jamshidi, Reiner Jung, Jóakim von Kistowski, Anne Koziolk, Johannes Kroß, Simon Spinner, Christian Vögele, Jürgen Walter, and Alexander Wert. *Performance-oriented devops: A research agenda*. Technical Report SPEC-RG-2015-01, SPEC Research Group - DevOps Performance Working Group, Standard Performance Evaluation Corporation (SPEC), 2015. <http://arxiv.org/abs/1508.04752>

- [8] Tomáš Martinec, Lukáš Marek, Antonín Steinhauser, Petr Tůma, Qais Noorshams, Andreas Rentschler, and Ralf Reussner. *Constructing performance model of JMS middleware platform*. In Proceedings of the 5th ACM/SPEC international conference on Performance engineering (ICPE '14). ACM, New York, NY, USA, 123-134. 2014. DOI: <https://ezproxy.itcr.ac.cr:2878/10.1145/2568088.2568096>
- [9] Mark Richards, Richard Monson-Haefel, David Chappell. *Java Message Service*. O'Reilly Media. Segunda Edición. 2009.
- [10] Nikolaus Huber, Steffen Becker, Christoph Rathfelder, Jochen Schweflinghaus, and Ralf H. Reussner. 2010. *Performance modeling in industry: a case study on storage virtualization*. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2 (ICSE '10), Vol. 2. ACM, New York, NY, USA, 1-10. DOI: <http://ezproxy.itcr.ac.cr:2075/10.1145/1810295.1810297>
- [11] Stephen Becker, Heiko Koziolk and Ralf Reussner. *The Palladio component model for model-driven prediction*. Journal of Systems and Software 82:3-22. Elsevier Science Inc. 2009. DOI: <http://dx.doi.org/10.1016/j.jss.2008.03.066>
- [12] Qais Noorshams. *Modeling and Prediction of I/O Performance in Virtualized Environments*. PhD thesis, Karlsruhe Institute of Technology (KIT), 2015. URL: <http://digbib.ubka.uni-karlsruhe.de/volltexte/1000046750>
- [13] Jens Happe, Holger Friedrich, Steffen Becker, and Ralf H. Reussner. *A pattern-based performance completion for Message-oriented Middleware*. In Proceedings of the 7th international workshop on Software and performance (WOSP '08). ACM, New York, NY, USA, 165-176. 2008. DOI: <http://ezproxy.itcr.ac.cr:2075/10.1145/1383559.1383581>
- [14] Heiko Koziolk. 2010. *Performance evaluation of component-based software systems: A survey*. Perform. Eval. 67, 8 (August 2010), 634-658. DOI: <http://ezproxy.itcr.ac.cr:2075/10.1016/j.peva.2009.07.007>
- [15] Thijmen de Gooijer, Anton Jansen, Heiko Koziolk, and Anne Koziolk. 2012. *An industrial case study of performance and cost design space exploration*. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE '12). ACM, New York, NY, USA, 205-216. DOI: <https://ezproxy.itcr.ac.cr:2878/10.1145/2188286.2188319>
- [16] Christoph Rathfelder, Steffen Becker, Klaus Krogmann and Ralf Reussner. *Workload-aware System Monitoring Using Performance Predictions Applied to a Large-scale E-Mail System*. 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, Helsinki, 2012, pp. 31-40. DOI: [10.1109/WICSA-ECSA.2012.11](https://doi.org/10.1109/WICSA-ECSA.2012.11).

- [17] Misha Strittmatter and Amine Kechaou. *The media store 3 case study system*. Technical Report 2016,1, Faculty of Informatics, Karlsruhe Institute of Technology, 2016. URL: <http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/3792054>
- [18] Murray Woodside, Greg Franks, and Dorina C. Petriu. *The Future of Software Performance Engineering*. Future of Software Engineering (FOSE '07), pages 171-187, May 2007. DOI 10.1109/FOSE.2007.32
- [19] Thijmen de Gooijer. *Performance Modeling of ASP.NET Web Service Applications: an Industrial Case Study*. Master's thesis, Malardalen University, Vasteras, Sweden, 2011.
- [20] Steffen Becker. *Palladio-Bench Screenshots*. <http://www.palladio-simulator.com/tools/screenshots/> Accesado el 20 de Marzo del 2018.
- [21] Yan Jin, Antony Tang, Jun Han, and Yan Liu. *Performance Evaluation and Prediction for Legacy Information Systems*. 29th International Conference on Software Engineering (ICSE'07), pages 540-549, May 2007.
- [22] Simonetta Balsamo, Antiniscia DiMarco, Paola Inverardi, and Marta Simeoni. *Model-based performance prediction in software development: A survey*. IEEE Trans. Softw. Eng., 30(5):295–310, May 2004.
- [23] Anne Kozirolek, Heiko Kozirolek, and Ralf Reussner. *PerOpteryx: automated application of tactics in multi-objective software architecture optimization*. In Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS (QoSA-ISARCS '11). ACM, New York, NY, USA, 33-42. 2011. DOI=<http://ezproxy.itcr.ac.cr:2075/10.1145/2000259.2000267>
- [24] Samuel Kounev and Fabian Brosig and Nikolaus Huber. *The Descartes Modeling Language*. Technical Report. Department of Computer Science, University of Wuerzburg, 2014.
- [25] Gunnar Brataas, Erlend Stav, Sebastian Lehrig, Steffen Becker, Goran Kopčak, and Darko Huljenic. 2013. *CloudScale: scalability management for cloud systems*. In Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE '13), Seetharami Seelam (Ed.). ACM, New York, NY, USA, 335-338. 2013. DOI: <https://ezproxy.itcr.ac.cr:2878/10.1145/2479871.2479920>