Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Programa de Maestría en Computación

Modelado y simulación de funciones en la nube en plataformas *Function-as-a-Service*

Propuesta de Tesis sometida a consideración del Departamento de Computación, para optar por el grado de Magíster Scientiae en Computación, con énfasis en Ciencias de la Computación

Estudiante

Carlos Martín Flores González

Profesor Asesor

Ignacio Trejos Zelaya

Noviembre, 2018

[git] • Branch: notas-reunion-cesar,@ 60c4154 • Release: (2018-10-08 23:25:07 -0600)

Commiter:Martin Flores, Head tags:(None) • References: (HEAD -> notas-reunion-cesar)

Indice

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Los servicios de funciones en la nube (*Function-as-a-Service, FaaS*) representan una nueva tendencia de la computación en la nube en donde se permite a los desarrolladores instalar código en una plataforma de servicios en la nube en forma de función y en donde la infraestructura de la plataforma es responsable de la ejecución, el aprovisionamiento de recursos, monitoreo y el escalamiento automático del entorno de ejecución. El uso de recursos generalmente se mide con una precisión de milisegundos y la facturación es por cada 100 ms de tiempo de CPU utilizado.

En este contexto, el "código en forma de función" es un código que es pequeño, sin estado, que trabaja bajo demanda y que tiene una sola responsabilidad funcional. Debido a que el desarrollador no se tiene que preocupar de los aspectos operacionales de la instalación o el mantenimiento del código, la industria empezó a describir este código como uno que no necesitaba de un servidor para su ejecución, o al menos de una instalación de servidor como las utilizadas en esquemas tradicionales de desarrollo, y acuñó el término *serverless* (sin servidor) para referirse a ello.

Serverless se utiliza entonces para describir un modelo de programación y una arquitectura en donde fragmentos de código son ejecutados en la nube sin ningún control sobre los recursos en donde el código se ejecuta. Esto de ninguna manera es una indicación de que no hay servidores, sino simplemente que el desarrollador delega la mayoría de aspectos operacionales al proveedor de servicios en la nube. A la versión de *serverless* que utiliza explícitamente funciones como unidad de instalación se le conoce como *Function-as-a-Service*[1].

Aunque el modelo FaaS brinda nuevas oportunidades, también introduce nuevos retos. Uno de ellos es el que tiene que ver con el rendimiento de la función, esto porque bajo en este modelo solamente se conoce una parte de la historia, la del código, pero se omiten los detalles de la infraestructura que lo ejecuta. La información de esta infraestructura, su configuración y capacidades es relevante para arquitectos y diseñadores de software para lograr estimar el comportamiento de una función en plataformas FaaS.

El problema de la estimación del rendimiento de aplicaciones en la nube como lo son las que se ejecutan en plataformas FaaS y arquitecturas basadas en microservicios es uno de los problemas que está recibiendo mayor atención especialmente dentro de la comunidad de investigación en ingeniería de rendimiento de software. Se argumenta que a pesar de la importancia de contar con niveles altos de rendimiento, todavía hay una falta de enfoques de ingeniería de rendimiento que tomen en cuenta de forma explícita las particularidades de los microservicios[2].

Si bien, para FaaS, existen plataformas *open source* por medio de las cuales se pueden obtener los detalles de la infraestructura y de esta manera lograr un mejor entendimiento acerca del rendimiento esperado, estas plataformas cuentan con arquitecturas grandes y complejas, lo cual hace que generar estimación

se convierta en una tarea sumamente retadora.

En este trabajo se plantea explorar la aplicación de modelado de rendimiento de software basado en componentes para funciones que se ejecutan en ambientes FaaS. Para esto se propone utilizar una función de referencia y a partir de esta, generar cargas de trabajo para recolectar datos de la bitácora(*logs*) de ejecución y extraer un modelo a partir de los mismos. Una vez que se cuente con un modelo, se procederá con su análisis y simulación con el fin de evaluar si el modelo generado logra explicar el comportamiento de la función bajo las cargas de trabajo utilizadas.

Capítulo 2

Antecedentes

2.1. Marco conceptual

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

Una definición comúnmente utilizada para definir ingeniería de rendimiento de software (*Software Performance Engineering* - SPE) es la que brinda en Woodside et al. [3]: *"Ingeniería de rendimiento de software representa toda la colección de actividades de ingeniería de software y análisis relacionados utilizados 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 puede 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. 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 [4] se deja la 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 son los que prevalencen en la industria[5] 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 remonta 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, *th-*

roughput y utilización de hardware para una configuración de un sistema bajo una carga de trabajo definida.

Ingeniería de rendimiento por medio de modelado

La importancia del modelado del rendimiento está motivada por el riesgo a que se presenten problemas graves de rendimiento y la creciente complejidad de sistemas modernos, lo que hace difícil abordar los problemas de rendimiento al nivel de código[6]. 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 un 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 [7] 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 comúnes de modelado incluyen redes de colas y también extensiones de estas como redes de colas en capas y varios tipos de redes de Petri, y procesos de álgebra estocástica.

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 implementación y ejecución en un ambiente de 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.[8]

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[5]. Los modelos de simulación también requieren usualmente de mayor tiempo de desarrollo que los modelos analíticos. En [3] 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".

2.1.2. Modelado y simulación de rendimiento basado en componentes

En este enfoque, modelos clásicos de rendimiento tales como redes de colas, redes de Petri estocásticas o procesos de álgebra estócastica son aplicados para modelar y analizar el rendimiento de software basado en componentes. La ingeniería de software basada en componentes se considera un sucesor del

desarrollo de software orientado a objetos, en esta los componentes de software son unidades de composición a las que se le define una especificación y una interfaz. A partir de estas, los arquitectos de software construyen e integran componentes de software entre sí, creando de esta manera sistemas más complejos.[4]

El reto en los modelos de rendimiento para componentes, es que el rendimiento de un componente de software en un sistema en ejecución depende del contexto en que está instlado y su perfil de uso, el cual es usualmente desconocido por el desarrollador del componente que creal el modelo de un componente individual.

Rendimiento de componentes de software

Szyperski[9] define un componente de software como: *"Un componente es una unidad de composición en aplicaciones de software, que posee un conjunto de interfaces y un conjunto de requisitos (una especificación), y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio"*.

Los componentes de software presentan los principios de ocultación de la información y la separación de responsabilidades. Fomentan la reutilización y preparan el sistema para que se puedan efectuar cambios de partes individuales. Permiten además una división de trabajo entre los desarrolladores de componentes y los arquitectos de software, lo que reduce la complejidad de la tarea de desarrollo. Los componentes de caja negra (*black-box*) solamente revelan sus interfases a los clientes, mientras que los componentes de caja blanca (*white-box*) permiten ver y modificar el código fuente de la implementanción

del componente como tal. Los componentes compuestos agrupan varios componentes en unidades más grandes.

Factores que influyen el rendimiento de un componente Especificar el rendimiento de componentes reutilizables es difícil porque esto no solamente va a depender de la implementación del componente, sino también del contexto en donde se encuentre instalado. Los factores que influyen en el rendimiento de los componentes son:

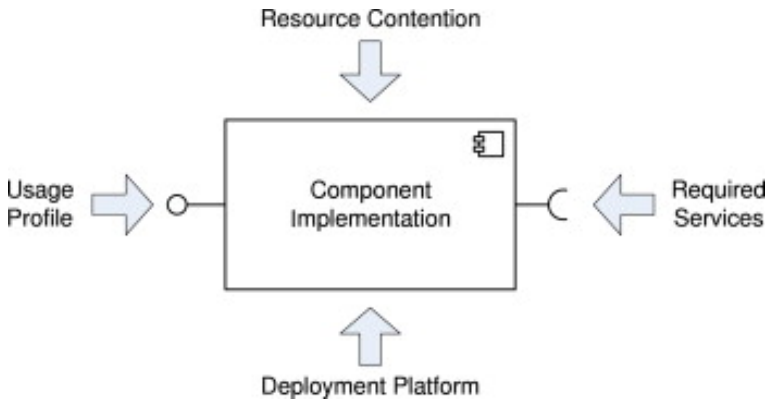

Figura 2.1: Factores que influyen en el rendimiento de un componente. Tomado de [4]

- *Implementación del componente*: los desarrolladores pueden implementar la funcionalidad especificada en una interfaz de diferentes formas. Dos componentes pueden proporcionar el mismo servicio pero presentar tiempos de ejecución diferentes cuando se ejecutan con los mismos recursos y con las mismas entradas.

- *Servicios requeridos*: cuando el componente A invoca el servicio del componente B, el tiempo de ejecución de B suma al tiempo de ejecución de A. Por lo tanto, el tiempo total de ejecución de un componente depende del tiempo de ejecución de los otros componentes/servicios que necesita.

- *Plataforma de instalación*: los arquitectos de software instalan un componente de software en diferentes plataformas. Una plataforma de instalación puede incluir varias capas de software (como por ejemplo contenedores de componentes o máquinar virtuales) y hardware (procesador, dispositivos de almacenamiento y red, etc)

- *Perfil de uso*: clientes pueden invocar los servicios del componente con diferentes parámetros de entrada. El tiempo de ejecución de un servicio puede cambiar dependiendo de los valores de lo parámetros de entrada.

- *Contención de recursos*: un componente de software típicamente no se ejecuta como un solo proceso aislado en una plataforma determinada. Los tiempos de espera inducidos para acceder a recursos limitados se suman al tiempo de ejecución de un componente de software.

Enfoques de ingeniería de rendimiento para software basado en componentes propuestos

La encuesta llevada a cabo en [4] 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 [10] 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 [4] durante los últimos diez años, los investigadores han propuesto muchos enfoques para evaluar el rendimiento (tiempos de respuesta, *throughput*, utlización de recursos) de sistemas de software basados en componentes. Estos enfoques abarcan tanto predicción como medición del rendi-

miento. 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 sequencia, 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 en-

foques 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 *middleware*.

- 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[11] es un enfoque de optimización de diseño de arquitecturas que manipula modelos especificados en *Palladio Component Model*[6] y utiliza el algoritmo evolutivo multiobjetivo `NSGA-II`. Para análisis de rendimiento utiliza redes de colas en capas. *Descartes Modeling Language*[12] 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*[1] [13] 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 *Palladio Component Model*.

[1] `http://www.cloudscale-project.eu/`

Modelado de Arquitecturas de Software con *Palladio Component Model*

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 2.2).

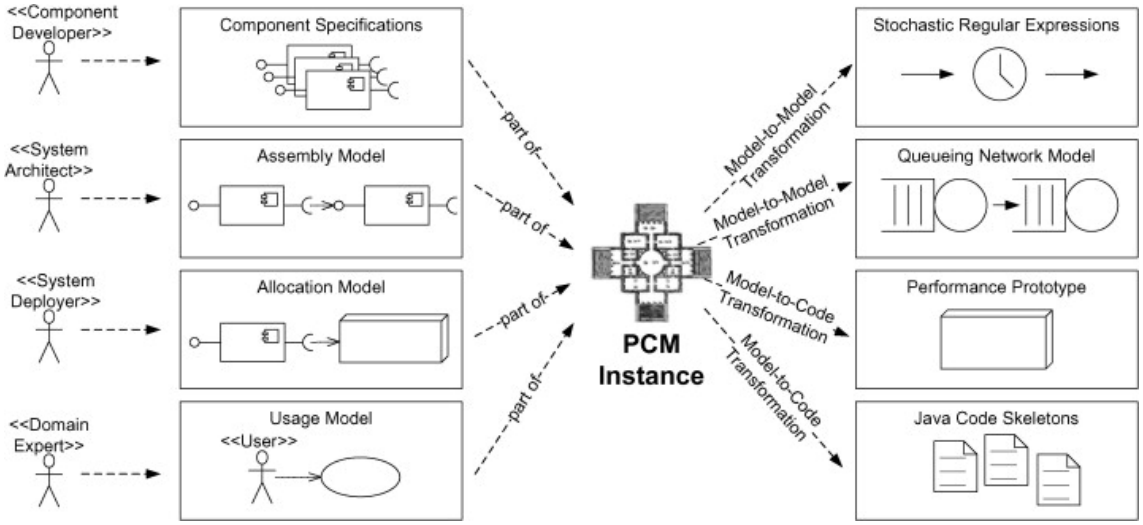

Figura 2.2: Instancia de un modelo PCM. Tomado de [14]

- **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 ins-

tancias 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 la instalación (*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 frencuencia 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[8].

Modelado de Arquitecturas de Software con *Descartes Modeling Language*

El *Descartes Modeling Language* (DML) es un lenguaje de modelado a nivel de arquitectura utilizado para describir calidad de servicio (*QoS* por sus siglas en inglés) y aspectos relacionados con la gestión de recursos de sistemas

de información dinámicos, infraestructuras y servicios. DML distingue explícitamente diferentes tipos de modelos que describen el sistema y sus procesos de adaptación desde un punto de vista técnico y lógico. Juntos, estos diferentes tipos de modelos forman una instancia DML. La idea detrás del uso de estos modelos es separar el conocimiento acerca de la arquitectura del sistema y el comportamiento de su rendimiento (aspectos técnicos) del conocimiento de los procesos de adaptación del sistema (aspectos lógicos)[12].

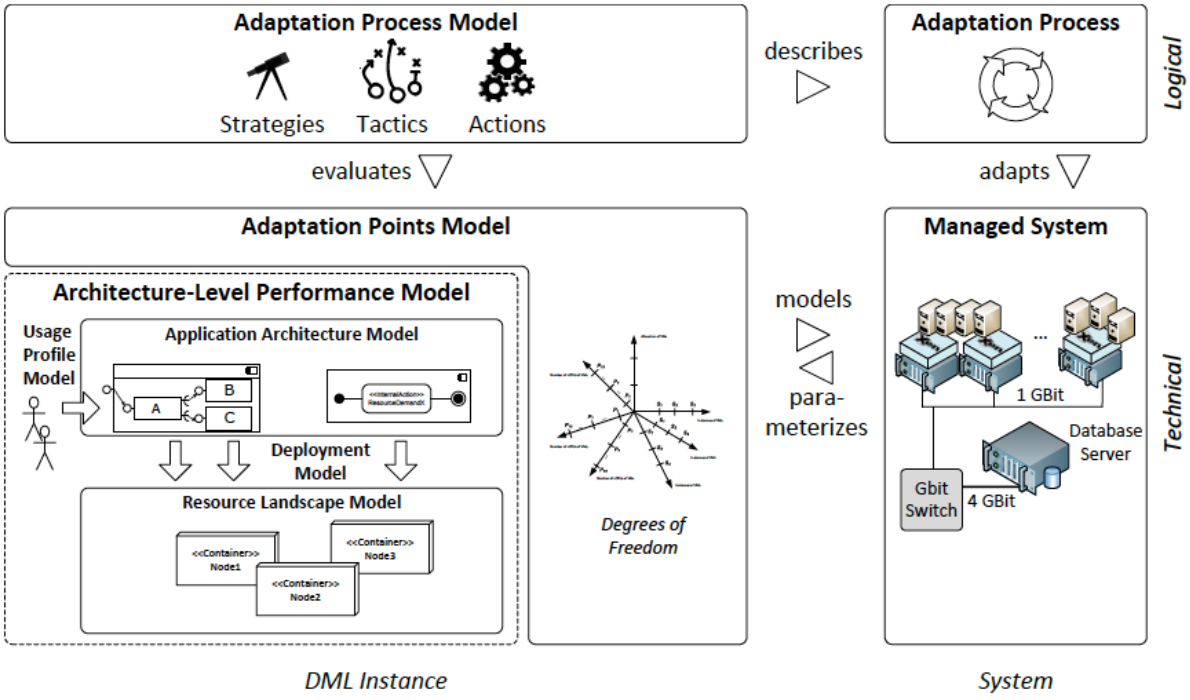

Figura 2.3: Relación de los diferentes modelos de una instancia DML y el sistema. Tomado de [12]

La figura 2.3 muestra la relación de los diferentes modelos que son parte de una instancia DML, el sistema gestionado (*managed system*) y el proceso de adaptación del sistema (*adaptation process*). En la esquina inferior derecha de la figura 2.3, se ve el sistema, el cual es gestionado por un proceso de adaptación, mostrado en la esquina superior derecha de la figura 2.3. En la esquina inferior izquierda de, se muestran modelos que reflejan los aspectos técnicos del sistema. Esos aspectos son los recursos de hardware y su distribución (*resource*

landscape model), los componentes de software y el comportamiento relevante al rendimiento (*application architecture model*), la instalación de componentes de software en el hardware (*deployment model*), el comportamiendo del uso y las cargas de trabajo de los usuarios del sistema (*usage profile model*) y los grados de libertad del sistema que pueden ser empleados para la adaptación del sistema en ejecución (*adaptation points model*). Por encima de estos modelos (esquina superior izquierda de la figura 2.3) se muestra el modelo de proceso de adaptación que especifica un proceso de adaptación que describe cómo el sistema se adapta a cambios en su ambiente. El proceso de adaptación aprovecha las técnicas de predicción de rendimiento en línea para razonar sobre posibles estrategias, tácticas y acciones de adaptación.

2.1.3. Computación en nube

La computación en la nube ha traído consigo nuevos estilos en la forma en cómo se desarrolla y se pone en producción el software. Esto motivado por ... **[EXTENDER]**

2.1.4. Trabajos relacionados

Ingeniería de rendimiento de software en aplicaciones en la nube

El estilo de arquitectura basado en microservicios es uno de los que ha logrado ganar mayor adopción y popularidad dentro de la comunidad de desarrolladores. Los microservicios, son una arquitectura de software que involucra la construcción y entrega de sistemas que se caracterizan por ser servicios pequeños, granulares, independientes y colaborativos [15]. Con respecto a SPE y

microservicios se reporta que existen muchos retos en investigación que han sido poco o nada abordados.

En [2] se planean el monitoreo, pruebas y modelado del rendimiento como las tres áreas en donde se carece de investigación y desarrollo de SPE y microservicios. Se argumenta que aún hace falta de enfoques de SPE que tomen en cuenta las particularidades de los microservicios. Aderaldo et al.[16] señala una falta de investigación empírica repetible sobre diseño, desarrollo y evaluación de aplicaciones de microservicios y que esto dificulta la evaluación de este tipo de aplicaciones ya que se cuenta con muy pocas aplicaciones y arquitecturas de referencia, así como de cargas de trabajo que contribuyan a caracterizar el comportamiento las mismas.

El reporte de [17] también proporciona un listado de los retos asociados con microservicios y SPE, haciendo énfasis en actividades de SPE relacionadas con la integración de actividades de desarrollo y de operaciones de puesta en producción y mantenimiento del software. Se argumenta que a pesar del alto nivel de adopción de prácticas de integración continua, entrega continua y DevOps de la comunidad de ingeniería de software, ninguna toma en cuenta aspectos relacionados con el rendimiento. Otros estudios, como el llevado a cabo en [18], indican que uno de los atributos de calidad que ha recibido mayor atención en la investigación en microservicios es el de la eficiencia del rendimiento pero vista mayoritariamente desde el punto de vista de la escalabilidad y mantenibilidad del código de los microservicios y su instalación.

[INCLUIR ESTUDIOS EN DONDE SI SE HA HECHO INVESTIGACION DE SPE Y MICROSERVICIOS?]

Serverless y Function-as-a-Service

FaaS es un área de lo que hoy se conoce como computación sin servidores o *serverless computing*. Amazon.com, uno de los principales propulsores de esta tendencia define *serverless* como una tecnología la cual permite construir y ejecutar aplicaciones y servicios sin necesidad de pensar en los servidores. Las aplicaciones *serverless* no requieren de aprovisionamiento, escalamiento y administración de ningún servidor. Se puede construir con ella casi cualquier tipo de aplicación o servicio, y todo lo que se requiere para ejecutar y escalar la aplicación con alta disponibilidad es gestionado por el proveedor del servicio[19].

FaaS se refiere a un tipo de aplicación *serverless* en donde la lógica del lado del servidor es escrita por un desarrollador pero, a diferencia de arquitecturas tradicionales, esta se ejecuta en contenedores de cómputo que no mantienen estado, son activados por medio de eventos, son efímeros (pueden durar solo una invocación) y son totalmente administrados por un tercero[20].

Cabe destacar que aunque el término *serverless* ha llegado a ser utilizado para referirse de forma directa a plataformas FaaS, hoy en día la aplicaciones de *serverless computing* abarcan otros tipos de servicios como almacenamiento, mensajería, análisis de datos, seguridad, entre otros.

Los investigadores han empezado a describir y analizar FaaS a través de encuestas y experimentos[1, 21, 22], y también por análisis económicos[23, 24], sin embargo se reporta que aún no se conoce mucho acerca de SPE en FaaS. En [2] se menciona que al igual que con microservicios, FaaS también requiere de nuevas estrategias de modelado para capturar el comportamiento del código bajo estas infraestructuras. Los modelos de rendimiento tradicionales basados en en la noción de máquinas independientes podría ser inadecuado.

En van Eyk et al.[25] se presenta un informe confeccionado por el *Standard Performance Evaluation Corporation RG Cloud Group*[2] (SPEC RG Cloud) sobre desafíos asociados a rendimiento en arquitecturas FaaS. Los principales temas son los que tienen que ver con evaluación y comparación de plataformas de FaaS, reducción del *overhead*, políticas de *scheduling*, la relación costo-rendimiento de una función y predicción del rendimiento. El informe también señala que actualmente muchas de las tareas de evaluación y pruebas para FaaS se vuelven complicadas porque no se cuenta con aplicaciones, arquitecturas ni cargas de trabajo de referencia, este es un trabajo que pretende abordar el SPEC RG Cloud. Con respecto a predicción del rendimiento, se indica que la aplicación de modelos de rendimiento de sistemas de software tradicionales en FaaS trae nuevos retos como la brecha de información (*information gap*) y el rendimiento de una función en particular. La brecha de información significa que el usuario de FaaS no está consciente de los recursos de hardware en los que las funciones son ejecutadas, mientras que por otro lado, la plataforma de FaaS no tiene información acerca de los detalles de la implementación de la función. Tal y como en las aplicaciones tradicionales, la entrada (tamaño, estructura y contenido) influyen en el rendimiento de una función, el hecho de tener una infraestructura oculta hace necesario encontrar nuevos modelos que logren predecir de forma precisa el rendimiento de una función. Técnicas de modelado desarrolladas para sistemas de software se podrían aprovechar para FaaS, como por ejemplo modelado y simulación de arquitecturas de software basados en componentes.

La aplicación de *serverless computing* es una área activa de desarrollo. En trabajos previos [26, 27] se han estudiado arquitecturas alternativas de *serverless computing* con el fin de explotar aspectos de rendimiento y/o abordar re-

[2] `https://research.spec.org/working-groups/rg-cloud.html`

tos técnicos que otras plataformas no han hecho. También se han investigado arquitecturas para recuperación de información[21] y *chatbots*[28] utilizando plataformas *serverless*. Muchas otras aplicaciones se han venido desarrollando en campos como *Machine Learning*, seguridad, Internet de las cosas, procesamiento de voz, sistemas de archivos, etc, y son solo una muestra del potencial de esta tecnología. Pese a este potencial, también se reporta que aún no se sabe mucho acerca de cuáles herramientas se usan para producir, instalar y ejecutar funciones[29]. En [22] se examinan factores qué pueden influir en el rendimiento de plataformas de *serverless computing*. De acuerdo a esto se logran identificar cuatro estado de una infraestructura *serverless*: *provider cold*, *VM cold*, *container cold*y *warm*. Además demuestra cómo el rendimiento de los servicios puede llegar a variar hasta 15 veces según estos estados.

Capítulo 3

Definición del Problema

En las plataformas FaaS en las que se ejecutan de funciones en la nube, la infraestructura tecnológica subyacente se oculta por completo de los desarrolladores y diseñadores. El conocimiento de la influencia de esta infraestructura y su configuración es vital para que los arquitectos de software puedan obtener predicciones significativas del comportamiento de una función pues, al omitirse la influencia que esta tiene, puede conducir a la generación de predicciones erróneas con respecto del rendimiento de una función. Una función que reporte tiempos de respuesta sumamente prolongados o bien la utilización de significativas cantidades de recursos puede generar grandes costos económicos y hasta llegar a ser rechazada por la plataforma de FaaS.

Las decisiones que hayan sido tomadas con poca información durante las etapas de diseño usualmente son muy difíciles de alterar y pueden impactar de forma negativa en los niveles requeridos de rendimiento de un sistema una vez que este ha sido puesto en producción. Es por esto que los arquitectos y diseñadores necesitan contar con la habilidad de predecir el rendimiento de una función trabajando a partir de diseños abstractos sin tener acceso a la imple-

mentación completa de la aplicación.

Capítulo 4

Justificación

4.1. Innovación

4.2. Impacto

4.3. Profundidad

Capítulo 5

Objetivos

5.1. Objetivo general

Diseñar un método para modelar funciones en la nube alojadas en plataformas de tipo *Function-as-a-Service* con el fin de evaluar los factores que pueden influir en su rendimiento.

5.2. Objetivos específicos

1. Revisar el estado del arte de trabajos relacionados con enfoques de predicción y medición del rendimiento en sistemas de software como servicio

2. Seleccionar un caso de uso de una función en la nube considerada como de referencia, con el propósito de analizar su comportamiento

3. Realizar experimentos sobre la función de referencia con herramientas de modelado y análisis de rendimiento de sistemas informáticos

4. Extraer un modelo de rendimiento a partir de los experimentos realizados

5. Validar experimentalmente el modelo planteado

6. Formular una guía metodológica

Capítulo 6

Alcance

Capítulo 7

Entregables

Capítulo 8

Metodología

Capítulo 9

Cronograma de actividades

Bibliografía

[1] I. Baldini, P. C. Castro, K. S. Chang, P. Cheng, S. J. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. M. Rabbah, A. Slominski, and P. Suter, "Serverless computing: Current trends and open problems," *CoRR*, vol. abs/1706.03178, 2017.

[2] R. Heinrich, A. van Hoorn, H. Knoche, F. Li, L. E. Lwakatare, C. Pahl, S. Schulte, and J. Wettinger, "Performance engineering for microservices: Research challenges and directions," in *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, ICPE '17 Companion, (New York, NY, USA), pp. 223–226, ACM, 2017.

[3] M. Woodside, G. Franks, and D. C. Petriu, "The future of software performance engineering," in *Future of Software Engineering (FOSE '07)*, pp. 171–187, May 2007.

[4] H. Koziolek, "Performance evaluation of component-based software systems: A survey," *Perform. Eval.*, vol. 67, pp. 634–658, Aug. 2010.

[5] T. de Gooijer, "Performance modeling of asp . net web service applications : an industrial case study," 2011.

[6] 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*. The MIT Press, 2016.

[7] Y. Jin, A. Tang, J. Han, and Y. Liu, "Performance evaluation and prediction for legacy information systems," in *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, (Washington, DC, USA), pp. 540–549, IEEE Computer Society, 2007.

[8] O.-Q. Noorshams, *Modeling and Prediction of I/O Performance in Virtualized Environments*. PhD thesis, 2015.

[9] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 2002.

[10] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, "Model-based performance prediction in software development: a survey," *IEEE Transactions on Software Engineering*, vol. 30, pp. 295–310, May 2004.

[11] A. Koziolek, H. Koziolek, and R. 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, (New York, NY, USA), pp. 33–42, ACM, 2011.

[12] S. Kounev, F. Brosig, and N. Huber, "The Descartes Modeling Language," tech. rep., Department of Computer Science, University of Wuerzburg, October 2014.

[13] G. Brataas, E. Stav, S. Lehrig, S. Becker, G. Kopčak, and D. Huljenic, "Clouds-cale: Scalability management for cloud systems," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, (New York, NY, USA), pp. 335–338, ACM, 2013.

[14] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *J. Syst. Softw.*, vol. 82, pp. 3–22, Jan. 2009.

[15] M. Cavallari and F. Tornieri, "Information systems architecture and organization in the era of microservices," in *Network, Smart and Open* (R. Lamboglia, A. Cardoni, R. P. Dameri, and D. Mancini, eds.), (Cham), pp. 165–177, Springer International Publishing, 2018.

[16] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, "Benchmark requirements for microservices architecture research," in *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*, pp. 8–13, May 2017.

[17] A. Brunnert, A. van Hoorn, F. Willnecker, A. Danciu, W. Hasselbring, C. Heger, N. R. Herbst, P. Jamshidi, R. Jung, J. von Kistowski, A. Koziolek, J. Kroß, S. Spinner, C. Vögele, J. Walter, and A. Wert, "Performance-oriented devops: A research agenda," *CoRR*, vol. abs/1508.04752, 2015.

[18] P. D. Francesco, I. Malavolta, and P. Lago, "Research on architecting microservices: Trends, focus, and potential for industrial adoption," in *2017 IEEE International Conference on Software Architecture (ICSA)*, pp. 21–30, April 2017.

[19] "Serverless computing - amazon web services," 2008. Obtenido el 26 Setiembre del 2018 de `https://aws.amazon.com/serverless`.

[20] M. Roberts, "Serverless architectures," 2018. Obtenido el 25 Setiembre del 2018 de `https://martinfowler.com/articles/serverless.html`.

[21] M. Crane and J. Lin, "An exploration of serverless architectures for information retrieval," in *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '17, (New York, NY, USA), pp. 241–244, ACM, 2017.

[22] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 159–169, April 2018.

[23] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang, "Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 179–182, May 2016.

[24] E. F. Boza, C. L. Abad, M. Villavicencio, S. Quimba, and J. A. Plaza, "Reserved, on demand or serverless: Model-based simulations for cloud budget planning," in *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*, pp. 1–6, Oct 2017.

[25] E. van Eyk, A. Iosup, C. L. Abad, J. Grohmann, and S. Eismann, "A spec rg cloud group's vision on the performance challenges of faas cloud architectures," in *Companion of the 2018 ACM/SPEC International Conference on*

Performance Engineering, ICPE '18, (New York, NY, USA), pp. 21–24, ACM, 2018.

[26] G. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 405–410, June 2017.

[27] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with openlambda," *Elastic*, vol. 60, p. 80, 2016.

[28] M. Yan, P. Castro, P. Cheng, and V. Ishakian, "Building a chatbot with serverless computing," in *Proceedings of the 1st International Workshop on Mashups of Things and APIs*, MOTA '16, (New York, NY, USA), pp. 5:1–5:4, ACM, 2016.

[29] J. Spillner, "Practical tooling for serverless computing," in *Proceedings of the10th International Conference on Utility and Cloud Computing*, UCC '17, (New York, NY, USA), pp. 185–186, ACM, 2017.