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*

Tesis 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

Mayo, 2019

```
 Git: (HEAD -> master)

Branch: master

Tag:

Release:

Commit: 5572359

Date: 2019-07-05 18:49:29 -0600

Author: Martin Flores

Email: martin.flores@bodybuilding.com

Committer: Martin Flores

Committer email: martin.flores@bodybuilding.com
```

Dedicatoria

Agradecimientos

Resumen

Abstract

Índice

1. Introducción **1**

2. Implementación de una *FaaS*: manejador de imágnenes **4**

2.1. *Manejador de imágenes* 6

2.1.1. Manejador de imágenes para SPE 7

2.2. Implementación del *manejador de imágenes* 10

2.2.1. Función Lambda: *Image-Handler* (IM-Simple) 11

Principales interacciones dentro de *Image-Handler* 16

2.2.2. Versiones alternas de *Image-Handler* 16

Versión instrumentalizada para Kieker y PMX (IM-KP) . . . 17

Versión instrumentalizada para AWS X-Ray (IM-XRay) . . . 20

2.3. Estrategia de extracción de modelo de rendmiento para *Image-Handler* 22

2.3.1. Modelo obtenido 23

Aporte de la versión IM-XRay a las simulaciones 25

2.4. Diseño Experimental 27

2.4.1. Utilizando *Image-Handler* para redimensionar imágenes de distintos tamaños 28

Invocaciones con imágenes menores a 500Kb 29

Análisis de resultados 33

Invocaciones con imágenes mayores a 500Kb y menores o igual a 1Mb 36

Análisis de resultados 42

Invocaciones con imágenes mayores a 1Mb y menores o igual a 2Mb 43

Análisis de resultados 50

Resultados Generales 52

2.4.2. Ejecución Secuencial Ininterrumpida de solicitudes de redimensionamiento 59

Ejecución de solicitudes de redimensionamiento simultáneas para imágenes de tamaño menor a 500Kb . 60

Ejecución de solicitudes de redimensionamiento simultáneas para imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb 62

Ejecución de invocaciones de redimensionamiento simultáneas para imágenes de tamaño mayor a 1Mb y menor o igual a 2Mb 64

Análisis de resultados 66

2.4.3. Variando el intervalo de las invocaciones de redimensionamiento 68

Estrategia de intervalo de invocaciones a *Image Handler* . 69

Ejecución de ráfagas de invocaciones de redimensionamiento para imágenes de tamaño menor a 500Kb 69

Variando el intervalo de las invocaciones de redimensionamiento en imágenes $\leq 500Kb$ 70

Variando el intervalo de las invocaciones de redimensionamiento en imágenes $500Kb \leq x \leq 1Mb$ 72

Variando el intervalo de las invocaciones de redimensionamiento en imágenes $1Mb \leq x \leq 2Mb$ 75

Análisis de resultados 76

Bibliografía **77**

Índice de figuras

2.1. Arquitectura del manejador de imágenes 5

2.2. Arquitectura del manejador de imágenes propuesto para el estudio 7

2.3. Carga de trabajo sugerida para el manejador de imágenes 8

2.4. Secuencia de acciones llevadas a cabo por *Image-Handler* 16

2.5. *Image-Handler* publicando eventos de rendimiento al servicio AWS X-Ray 22

2.6. Publicando mediciones del rendimiento de la función Lambda. . 25

2.7. Convirtiendo una bitácora de Kieker a una instancia de PCM por medio de PMX. 26

2.8. Distribución del tamaño de imágenes $\leq 500Kb$ 30

2.9. Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $\leq 500Kb$ en IM-Simple 31

2.10. Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $\leq 500Kb$ en las simulaciones de *Palladio Workbench* 32

2.11. IM-Simple *vs* simulaciones en PCM: 1000 solicitudes de redimensionamiento de imágenes de tamaño $\leq 500Kb$. 34

2.12. Probabilidad acumulada en solicitudes de redimensionamiento en imágenes de tamaño $\leq 500Kb$ en *Palladio Workbench* 35

2.13. Distribución del tamaño de imágenes $500Kb \leq x \leq 1Mb$ 38

2.14. Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $500Kb \leq x \leq 1Mb$ en IM-Simple 39

2.15. Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $500Kb \leq x \leq 1Mb$ en las simulaciones de Palladio 41

2.16. Solicitudes de redimensionamiento de imágenes de tamaño $500Kb \leq x \leq 1Mb$ 42

2.17. Probabilidad acumulada de solicitudes de redimensionamiento en imágenes $500Kb \leq x \leq 1Mb$ en PCM 44

2.18. Distribución del tamaño de imágenes de tamaño $1Mb \leq x \leq 2Mb$ 46

2.19. Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $1Mb \leq x \leq 2Mb$ en IM-Simple 47

2.20. Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $1Mb \leq x \leq 2Mb$ en las simulaciones de Palladio 49

2.21. Solicitudes de redimensionamiento de imágenes de tamaño $1Mb \leq x \leq 2Mb$ 50

2.22. Probabilidad acumulada de solicitudes de redimensionamiento en imágenes $1Mb \leq x \leq 2Mb$ en PCM 53

2.23. 3000 Simulaciones: Función de probabilidad acumulada para los tres escenarios de pruebas 54

2.24. *Image-Handler* publicando eventos de rendimiento al servicio AWS X-Ray 58

2.25. hola 61

2.26. 1000 solicitudes de redimensionamiento en imágenes $500Kb \leq x \leq 1Mb$. La línea azul representa las solicitudes hechas utilizando imágenes aleatorias. La línea roja, las solicitudes utilizando una única imagen. 63

2.27. 1000 solicitudes de redimensionamiento en imágenes $1Mb \leq x \leq 2Mb$. La línea azul representa las solicitudes hechas utilizando imágenes aleatorias. La línea roja, las solicitudes utilizando una única imagen. 65

2.28. El espacio delimitado por la línea punteada roja representa 10 minutos de inactividad entre la primer ráfaga y la segunda. El segundo espacio delimitado por la línea punteada roja representa 20 minutos de inactividad entre la segunda y la tercera ráfaga. . . 71

2.29. El espacio delimitado por la línea punteada roja representa 10 minutos de inactividad entre la primer ráfaga y la segunda. El espacio delimitado por la línea punteada verde representa 20 minutos de inactividad entre la segunda y la tercera ráfaga. 73

2.30. El espacio delimitado por la línea punteada roja representa 10 minutos de inactividad entre la primer ráfaga y la segunda. El espacio delimitado por la línea punteada verde representa 20 minutos de inactividad entre la segunda y la tercera ráfaga. 75

Lista de Tablas

2.1. Resumen de datos estadísticos 33

2.2. Resumen de datos estadísticos 41

2.3. Resumen de datos estadísticos 51

2.4. Resumen de datos estadísticos de los tres experimentos propuestos 57

2.5. Tiempo de respuesta de la primer invocación de redimensionamiento en cada ráfaga. 74

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 forma de función, en una plataforma de servicios en la nube 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 usualmente 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 necesita preocuparse 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 de cómputo 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 tiene que ver con el rendimiento de la función, puesto que 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 consideren 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 cuen-

tan 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 ellos. Una vez que se cuente con un modelo, se procederá con su análisis y simulación a fin de evaluar si el modelo generado logra explicar el comportamiento de la función bajo las cargas de trabajo utilizadas.

Esta propuesta está organizada de la siguiente manera: en el capítulo **??** se presenta un marco conceptual sobre ingeniería de rendimiento de software y trabajos de investigación relacionados con ingeniería de rendimiento de software en aplicaciones en la nube, microservicios y *serverless*. En el capítulo **??** se define el problema a resolver. En el capítulo **??** se proporciona una justificación del proyecto desde las perspectivas de innovación, impacto y profundidad. El objetivo general y los objetivos específicos se plantean en el capítulo **??**. El alcance del proyecto se define en el capitulo **??**. Los entregables que se generarán a partir de esta propuesta se listan en el capítulo **??**. La metodología de trabajo se indica en el capítulo **??**. La propuesta concluye en el capítulo **??**, donde se presenta el cronograma de actividades.

Capítulo 2

Implementación de una función en la nube: manejador de imágenes

Uno de los principales problemas de hacer ingeniería de rendimiento para software en la nube es que no existen aplicaciones de referencia que hayan ganado popularidad o cuyo desarrollo se encuentre activo. A pesar de esto y de su reciente adopción, la industria ha empezado a reconocer casos de uso en donde las aplicaciones *serverless* encajan mejor. Amazon Web Services(AWS)[3] reconoce cinco patrones de uso predominantes en su servicio AWS Lambda:

1. Procesamiento de datos dirigidos por eventos.
2. Aplicaciones Web.
3. Aplicaciones móviles e Internet las cosas (IoT).
4. Ecosistemas de aplicaciones *serverless*.
5. Flujos de trabajo dirigidos por eventos.

Uno de las aplicaciones más comunes en *serverless* es desencadenar acciones luego de que ocurre un evento (1), por ejemplo luego de la modificación de un registro en una base de datos o bien luego de que se publica un mensaje en una cola de mensajería. Esto puede provocar que se active una función Lambda[1] que toma como entrada el evento recién publicado para su posterior procesamiento. Este estilo de caso de uso encaja bien en ambientes híbridos: ambientes en donde tecnologías *serverless* se aprovechan para realizar funciones específicas dentro de una aplicación (o aplicaciones) más grande.

AWS ha publicado una serie de arquitecturas de referencia[4] para su plataforma FaaS, AWS Lambda. Dentro de estas arquitecturas se destaca el caso de uso de un manejador de imágenes (*Image Handler*)[5].

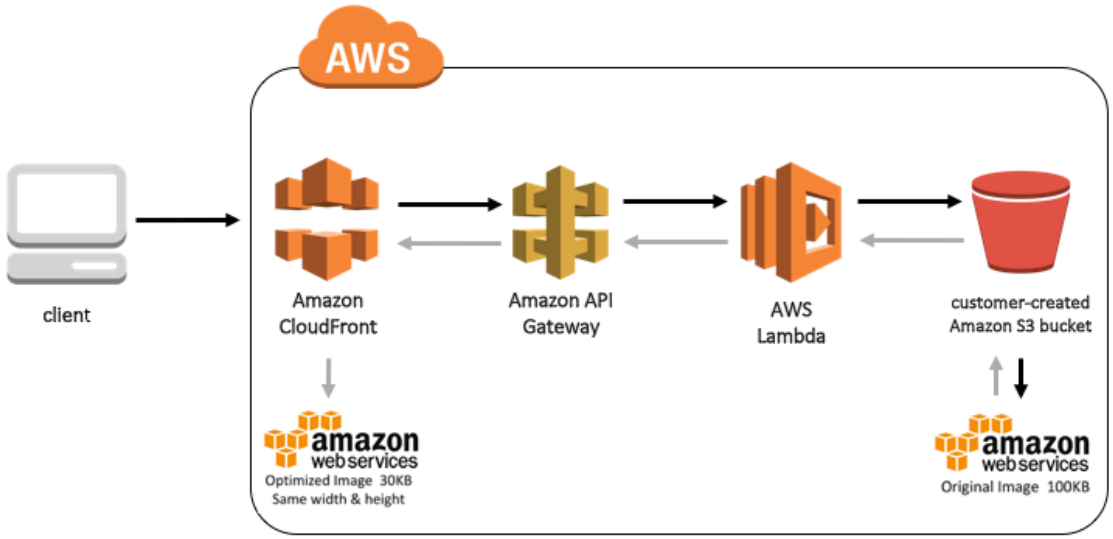

Figura 2.1: Arquitectura del manejador de imágenes. Tomado de [5]

[1] En la plataforma AWS Lambda

2.1. *Manejador de imágenes*

Sitios Web con imágenes grandes pueden experimentar tiempos de carga prolongados, es por esto que los desarrolladores proporcionan diferentes versiones de cada imagen para que se acomoden a distintos anchos de banda o diseños de página. Para brindar tiempos de respuesta cortos y disminuir el costo de la optimización, manipulación y procesamiento de las imágenes, AWS propone un manejador de imágenes *serverless*, al cual se le pueda delegar tal trabajo como una función Lambda sobre la plataforma FaaS.

A continuación se describe la arquitectura de la figura 2.1:

1. Amazon CloudFront provee una capa de *cache* para reducir el costo del procesamiento de la imagen

2. Amazon API Gateway brinda acceso por medio de HTTP a las funciones Lambda

3. AWS Lambda obtiene la imagen de un repositorio de Amazon Simple Storage Service (Amazon S3) y por medio de la implementación de la función se retorna una versión modificada de la imagen al API Gateway

4. El API Gateway retorna una nueva imagen a CloudFront para su posterior entrega a los usuarios finales

Cabe mencionar que, en este contexto, una versión modificada de una imagen será cualquier imagen que haya presentado algún tipo de alteración con respeto de una imagen original como, por ejemplo, cambios de tamaño, color, metadatos, etc.

2.1.1. Manejador de imágenes para SPE

Para este estudio se proponemos implementar una variación del manejador de imágenes de la sección 2.1, que se muestra en la figura 2.2.

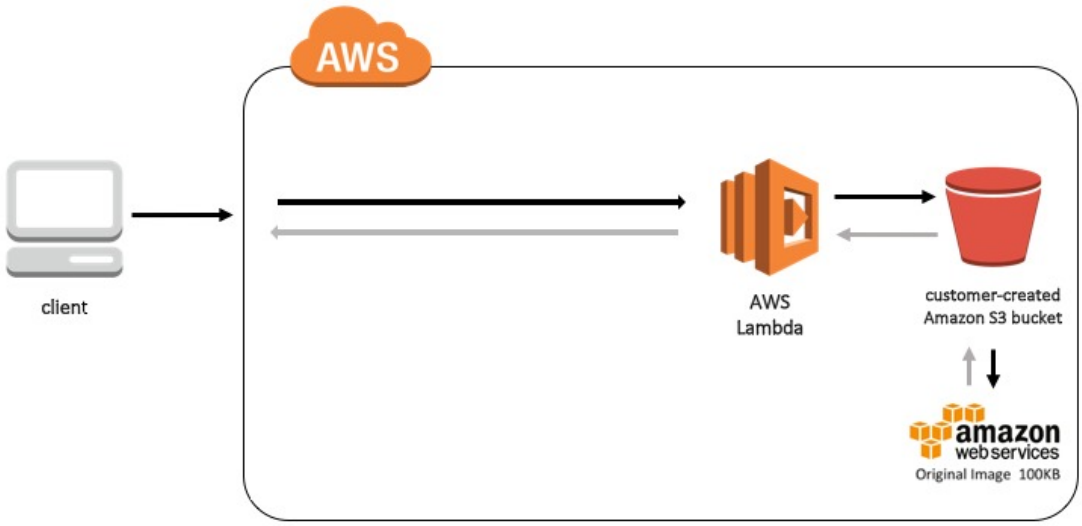

Figura 2.2: Arquitectura del manejador de imágenes propuesto para el estudio.

Se han dejado por fuera intencionalmente el AWS CloudFront y el AWS API Gateway. La razón de esto es porque se pretende ejercitar la función Lambda directamente. Se implementará una función Lambda que entregue a partir de una solicitud de redimensionamiento de una imagen almacenada, otra con dimensiones diferentes producida "al vuelo" como respuesta a la solicitud. Por ejemplo, si la imagen original mide 500 pixeles de ancho y alto, entregar una con dimensiones de 100 pixeles de ancho y alto.

Las actividades involucradas en el proceso de redimensionamientos de imágenes se muestran en la figura 2.3

1. Se envía una solicitud de redimensionamiento de imagen en formato `JSON` a la función Lambda con los datos acerca de la localización de la imagen y

su nuevo tamaño.

2. La solicitud de redimensionamiento llega a la función Lambda.

3. La función Lambda solicita al servicio de almacemiento AWS S3 la imagen.

4. AWS S3 entrega a la función Lambda la imagen solicitada.

5. La función Lambda inicia el redimensionamiento de la imagen de acuerdo a los parámetros solicitados.

6. La nueva imagen modificada se entrega al cliente(s).

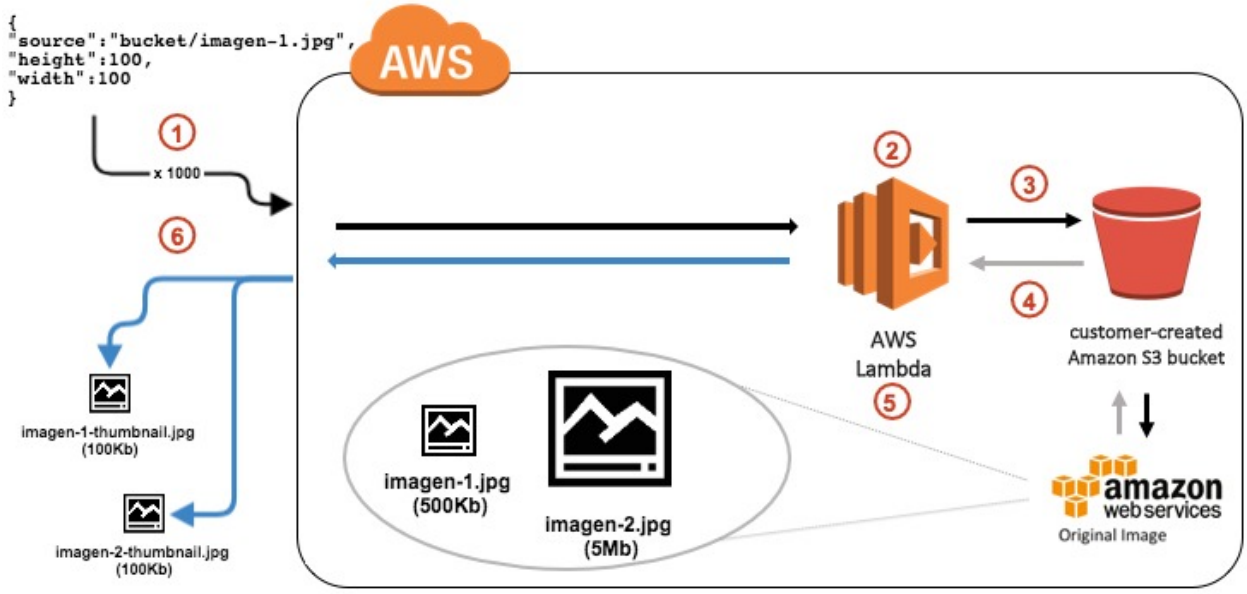

Figura 2.3: Carga de trabajo sugerida para el manejador de imágenes

A la función Lambda se le realizarán pruebas con imágenes de entrada de distinto tamaño y cargas de trabajo variables para evaluar su comportamiento bajo estos escenarios. Se desea observar el impacto de las pruebas en el tiempo de respuesta de la función. Los resultados obtenidos a partir de estas pruebas van a servir como un punto de referencia para experimentos futuros, como los

que se indican en la Sección **??**. La figura 2.3 muestra una sugerencia de dos posibles cargas de trabajo:

1. 100 solicitudes de cambio de tamaño de una imagen grande. En la figura 2.3, `imagen-2.jpg` de tamaño de 5Mb, representa una imagen grande.

2. 100 solicitudes de cambio de tamaño de una imagen pequeña. En la figura 2.3, `imagen-1.jpg` de tamaño menor o igual a 500Kb, representa una imagen pequeña.

En principio las cargas de trabajo generadas serían *cerradas*, lo que quiere decir que una solicitud se ejecuta solamente hasta que la anterior se termina. Esto ayudará en principio a tener mejor trazabilidad de lo que ocurre con la función.

¿Por qué este caso de uso se considera relevante? A continuación se listan las características que hacen este caso de uso representativo e interesante:

- Sencillo de entender e implementar: se cuenta únicamente con una función la cual lleva a cabo una tarea muy específica.

- Popular: sigue un patrón de procesamiento dirigido por eventos y, como se señala en [3], este es uno de los más populares que se ha empezado a adoptar para aplicaciones *serverless*. Otra de las razones de la popularidad de este caso de uso es que permite a los desarrolladores crear una unidad de instalación independiente y especializada para el manejo de imágenes, liberando así a sus servidores y aplicaciones del manejo de las peticiones y lógica asociadas a estas.

- Replicable en otros proveedores de servicios en la nube: varias de las arquitecturas de referencia para *serverless* propuestas por Amazon, están compuestas por herramientas y servicios muy propios de su plataforma, lo cual hace muy difícil su reproducibilidad utilizando otros proveedores. Aunque en principio este trabajo plantea ser elaborado en la plataforma FaaS de Amazon Web Services, AWS Lambda, otros proveedores de servicios (ver sección **??**) en la nube cuentan con sus propias plataformas de FaaS y de almacenamiento, lo cual permitiría replicar lo aquí propuesto en ellos.

- Replicable en los lenguajes de programación soportados por plataformas FaaS: actualmente JavaScript, Java (y lenguajes basados en la *Java Virtual Machine*), Python, C# y Go son los principales lenguajes de programación soportados por las plataformas FaaS. El caso de uso propuesto, no presenta ningún tipo de característica que lo ate a un lenguaje de programación en particular. En todos ellos se cuentan con bibliotecas para manejo de imágenes tanto de forma nativa como por medio de soluciones de terceros.

2.2. Implementación del *manejador de imágenes*

Existen soluciones disponibles que se pueden estudiar para implementar un manejador de imágenes. Amazon proporciona dos ejemplos que siguen la arquitectura de la figura 2.1:

1. **serverless-image-resizing**[2]: escrita en lenguaje JavaScript. Utiliza el mo-

[2] `https://github.com/amazon-archives/serverless-image-resizing`

dulo *sharp*[3] de NodeJS para aplicar operaciones de conversión en imágenes tales como redimensionamiento, rotación y corrección gamma.

2. **serverless-image-handler**[4]: escrita en lenguaje Python. Hace uso del paquete *Thumbor*[5] de código abierto para realizar operaciones de redimensionamiento, rotación, recorte y aplicación de filtros en imágenes.

A pesar que Amazon recomienda el uso de *serverless-image-handler* sobre *serverless-image-resizing*, ambas soluciones siguen un patrón sumamente similar en su codificación e instalación.

Otro ejemplo de una función en la nube encargada de ofrecer un servicio de redimensionamiento en imágenes, es la *Course_LambdaResizer*, una función lambda usada como referencia en el curso *"Serverless API on AWS for Java developers"* ofrecido en el sitio Web Udemy[6]. Esta función está escrita en lenguaje Java y utiliza la biblioteca *imgscalr*[7] para redimensionar imágenes.

Para este estudio, se implementó una función escrita en lenguaje Java. Esto motivado principalmente por la compatibilidad de este lenguaje con las herramientas para monitoreo de aplicaciones y extracción de modelos de rendimiento, Kieker y PMX respectivamente.

2.2.1. Función Lambda: *Image-Handler* (IM-Simple)

La función Lambda creada para este estudio lleva por nombre *Image-Handler*. El código fuente y documentación relacionada con la misma se encuentra dis-

[3] https://github.com/lovell/sharp
[4] https://github.com/awslabs/serverless-image-handler
[5] http://thumbor.org
[6] https://www.udemy.com/serverless-api-aws-lambda-for-java-developers
[7] https://github.com/rkalla/imgscalr

ponible en GitHub.com, en el repositorio de código: `https://github.com/seminario-dos/image-handler`. El punto de entrada de la función Lambda es la clase `ImageHandler.java`. Esta función se encarga de realizar tres operaciones para procesar una solicitud de redimensionamiento de imagen:

1. Procesar la solicitud de redimiensionamiento (la entrada) que viene dada en formato JSON. Esta solicitud de redimensionamiento contiene entre otras cosas:
   - El nombre de la imagen original que reside en el servicio Amazon S3.
   - Los parámetros de altura y ancho a los que se desea redimensionar la imagen original.
2. Obtener la imagen del servicio Amazon S3 y posteriormente aplicar la operación de redimiensionamiento sobre la misma de acuerdo a los parámetros de altura y ancho especificados en la solicitud de redimensionamiento.
3. Tomar la imagen redimensionada, codificarla en `Base64` y escribir el resultado en el flujo(*stream*) de salida de la función Lambda.

Un extracto de la clase `ImageHandler.java` se muestra en el listado 2.1. En la línea `22` se procesa el evento de entrada que viene dado en formato JSON. Como resultado de esto se entrega un objeto `ImageRequest` el cual contiene la información de la solicitud de la imagen que se desea redimensionar y que se encuentra alojada en el servicio Amazon S3.

En la línea `24` se llama al servicio `ImageService` con el fin de obtener la imagen original (de acuerdo a la información presente en el `ImageRequest` proporcionado) y se aplica la operación de redimensionamiento.

Por último, en la línea 26, `ImageHandlerResponseWriter.writeResponse()` toma la nueva imagen, con nuevas dimensiones de alto y ancho, la codifica en `Base64` y escribe el resultado en el *stream* de salida de la función.

```
1  public class ImageHandler implements RequestStreamHandler {
2
3      private static final AppConfig APP_CONFIG;
4      private final AppConfig appConfig;
5
6      static {
7          APP_CONFIG = AppConfig.getInstance();
8      }
9
10     public ImageHandler() {
11         this(APP_CONFIG);
12     }
13
14     public ImageHandler(AppConfig appConfig) {
15         this.appConfig = appConfig;
16     }
17
18     @Override
19     public void handleRequest(InputStream inputStream,
20                               OutputStream outputStream,
21                               Context context) throws IOException {
22         ImageRequest imageRequest =
23             this.inputEventParser().processInputEvent(inputStream);
24         InputStream imageResized =
25             this.imageService().getImageFrom(imageRequest);
26         this.imageHandlerResponseWriter()
27             .writeResponse(imageResized, outputStream, imageRequest);
28     }
29
30     private InputEventParser inputEventParser() {
31         return this.appConfig.getInputEventParser();
32     }
33
34     private ImageService imageService() {
35         return this.appConfig.getImageService();
36     }
37
38     private ImageHandlerResponseWriter imageHandlerResponseWriter() {
39         return this.appConfig.getImageHandlerResponseWriter();
40     }
41 }
```

Listing 2.1: Clase `ImageHandler.java`

Las funciones Lambda en AWS reciben como entrada un objeto JSON. Este objeto puede contener distintos campos dependiendo del servicio que haya

invocado previamente la ejecución de la función Lambda. Debido a que la función *Image-Handler* pretende ser invocada por medio de solicitudes `HTTP`, esta se configuró para que trabajara en conjunto con el servicio `API Gateway`. Dentro de este servicio se creó un recurso Web que entrega solicitudes de tipo `HTTP GET` a la función Lambda para su posterior procesamiento.

En términos generales, cada vez que una solicitud `HTTP GET` ingresa al `API Gateway` con el siguiente formato:
`https://{host}/image/{image}?width={value}&height={value}`
se tomarán el nombre de la imagen original que viene en el parámetro `image` y los parámetros de ancho y alto, `width` y `height` respectivamente, y se pasarán como parámetros de entrada a la función Lambda como parte de un objeto JSON. Este objeto JSON contiene otros campos que dan a conocer a la función Lambda información acerca de la solicitud `HTTP`.

Ejemplo: para la siguiente solicitud `HTTP`:

```
GET https://{host}/images/original-pic.jpg?width=50&height=66
```

`API Gateway` produce el objeto JSON listado en 2.2. A pesar que el objeto JSON incluye otros campos, para efectos del *Image-Handler* solamente tres de ellos serán utilizados:

1. `pathParameters`: contiene el nombre de la imagen original a ser redimensionada.

2. `isBase64Encoded`: señala si la solicitud necesita ser codificada en `Base64` o no.

3. `queryStringParameters`: bajo esta propiedad se listan los parámetros de ancho(`width`) y alto(`height`).

```
{
  "headers": {
    "Accept": "*/*",
    "User-Agent": "HTTPie/1.0.2",
    "Connection": "keep-alive",
    "X-Forwarded-Proto": "http",
    "Host": "localhost:3000",
    "Accept-Encoding": "gzip, deflate",
    "X-Forwarded-Port": "3000"
  },
  "pathParameters": {
    "image": "original-pic.jpg"
  },
  "path": "/images/original-pic.jpg",
  "isBase64Encoded": true,
  "requestContext": {
    "accountId": "123456789012",
    "path": "/images/{image+}",
    "resourceId": "123456",
    "stage": "prod",
    "requestId": "c6af9ac6-7b61-11e6-9a41-93e8deadbeef",
    "identity": {
      "cognitoIdentityPoolId": null,
      "accountId": null,
      "caller": null,
      "apiKey": null,
      "sourceIp": "127.0.0.1",
      "cognitoAuthenticationType": null,
      "cognitoAuthenticationProvider": null,
      "userArn": null,
      "userAgent": "Custom User Agent String",
      "user": null
    },
    "resourcePath": "/images/{image+}",
    "httpMethod": "GET",
    "extendedRequestId": null,
    "apiId": "1234567890"
  },
  "resource": "/images/{image+}",
  "httpMethod": "GET",
  "body": null,
  "queryStringParameters": {
    "width": "50",
    "height": "66"
  },
  "stageVariables": null
}
```

Listing 2.2: Clase `ImageHandler.java`

Principales interacciones dentro de *Image-Handler*

La figura 2.4 muestra las principales interacciones que lleva a cabo la función *Image-Handler*. Tanto las acciones como los actores involucrados, concuerdan con lo descrito en la Sección 2.2.1 aunque, a diferencia de lo descrito allí, aquí se presenta la clase `S3Dao` que es la que se encarga de buscar y traer la imagen original del servicio Amazon S3.

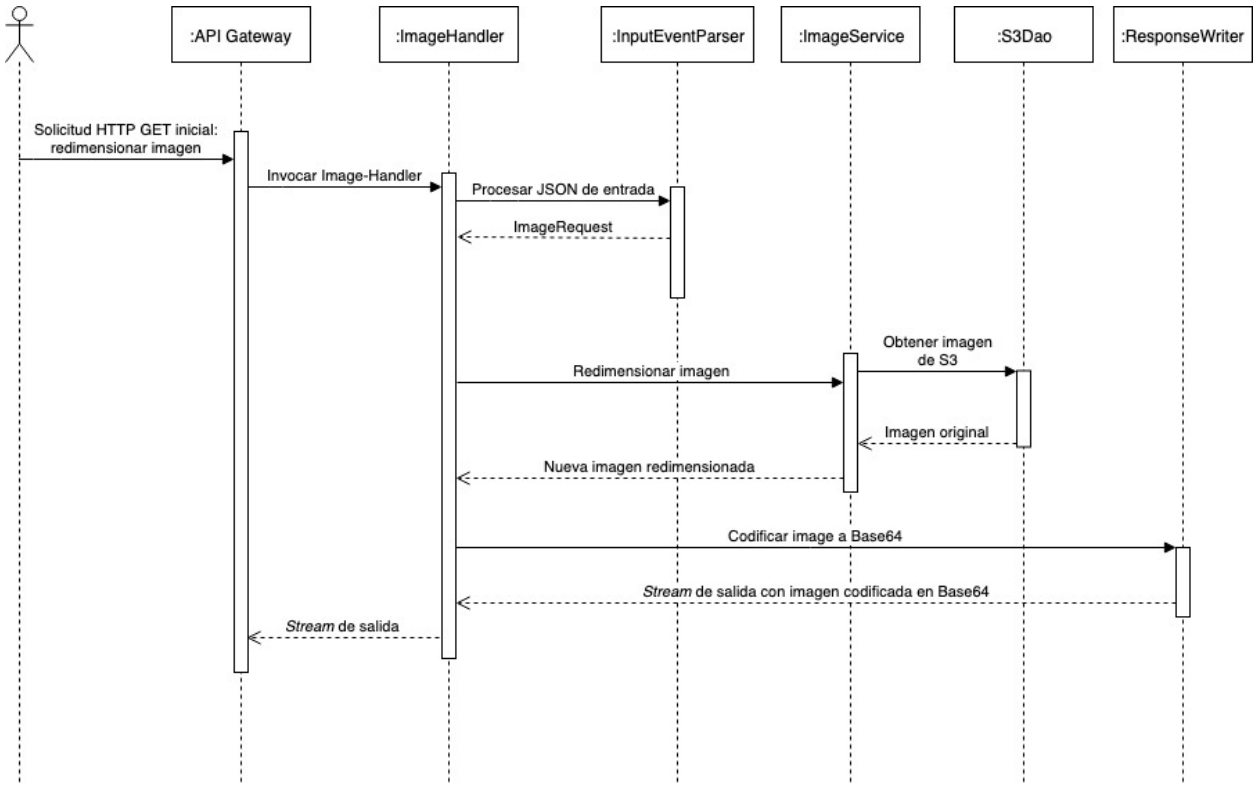

Figura 2.4: Secuencia de acciones llevadas a cabo por *Image-Handler*

2.2.2. Versiones alternas de *Image-Handler*

Aparte de la versión original de *Image-Handler*, se crearon dos versiones alternas con el fin de instrumentalizar el código fuente para la extracción y generación de un modelo en PCM a partir, y para la obtención de mediciones del

rendimiento.

En la primer version, el código se modificó para generar rastros del rendimiento de la función Lambda y extraer a partir de estos un modelo de rendimiento en PCM. La segunda versión fue modificada para generar rastros de rendimiento pero, a diferencia de la versión anterior, los datos obtenidos en esta versión fueron de mayor utilidad para afinar las estimaciones de rendimiento de los componentes involucrados en la función.

Versión instrumentalizada para Kieker y PMX (IM-KP)

Uno de los principales objetivos de este trabajo es el de obtener un modelo de rendimiento a partir del código en ejecución de una función en la nube. A pesar que, el modelo de rendimiento puede ser creado por los diseñadores e implementadores sin necesidad de una herramienta que ayude a su extracción, el uso de una herramienta especializada para esto contribuye a la generación de un modelo que pueda incluir mayores niveles de detalle en cuanto a la estructura y estimaciones del comportamiento de un software. Además, debido a que se están dando los primeros pasos en el campo del modelado y simulación de rendimiento basado en componentes, es preferible delegar tareas de extracción y estimaciones a una herramienta(s) con el fin de aprender de lo resultados obtenidos e ir introduciendo cambios paulatinamente; la creación manual de modelos de rendimiento puede llegar a ser muy compleja, consumir mucho tiempo y ser propensa a errores.

Para lograr esto, se seleccionaron dos herramientas, Kieker y PMX, las cuales en conjunto proporcionan un marco de trabajo por medio del cual se puede obtener mediciones del rendimiento de una aplicación y luego, a partir de estas,

extraer un modelo de rendimiento basado en PCM. La selección de estas herramientas y el enfoque de medición y extracción de modelos de rendimiento se seleccionó luego de estudiar el *enfoque de ingeniería de rendimiento declarativo* propuesto en [6].

Kieker se utiliza para el estudiar el comportamiento del rendimiento del sistema a partir de las entradas/rastros que se registran en una bitácora. Kieker ofrece adaptadores de monitoreo (*monitoring adapters*) escritos en lenguaje Java (también ofrece adaptadores en otros lenguajes). Los dos principales componentes de Kieker son: `Kieker.Monitoring` y `Kieker.Analysis`. `Kieker.Monitoring` es el responsable de la instrumentación del código, recolección de datos y registro(*logging*). El componente `Kieker.Analysis` es el responsable de leer, analizar y visualizar los datos monitoreados.

En esta versión de *Image-Handler* se modificó el código original para generar registros del rendimiento de la ejecución de la función utilizando las bibliotecas proporcionadas por Kieker, utilizando como referencia lo especificado en el manual de usuario de Kieker[7]. Se crean objetos de tipo `OperationExecutionRecord` los cuales son los que contienen la información acerca del rendimiento de una invocación sobre alguna parte del código. En el listado 2.3 se puede ver un extracto del código de la función instrumentalizada para que genere objetos `OperationExecutionRecord`.

Adicionalmente se configuró la biblioteca para que publique los objetos `OperationExecutionRecord` a modo eventos a una cola de mensajería *Java Message Service* (JMS), en lugar de una bitácora local.

```
1 public class ImageHandlerKieker implements RequestStreamHandler {
2     private static final IMonitoringController MONITORING_CONTROLLER;
3     static {
4         MONITORING_CONTROLLER = MonitoringController.getInstance();
5     }
6     .
```

```
7       .
8       .
9
10      @Override
11      public void handleRequest(InputStream inputStream, OutputStream
        outputStream, Context context) throws IOException {
12
13          final long tin = MONITORING_CONTROLLER.getTimeSource().getTime
        ();
14          handleRequestInternal(inputStream, outputStream, context);
15          final long tout = MONITORING_CONTROLLER.getTimeSource().getTime
        ();
16          final OperationExecutionRecord e = new OperationExecutionRecord
        ("public void "+ this.getClass().getName()+".handleRequest(
        InputStream, OutputStream, Context)",
17                      OperationExecutionRecord.NO_SESSION_ID,
18                      OperationExecutionRecord.NO_TRACE_ID,
19                      tin, tout,
20                      InetAddress.getLocalHost().getHostName(),
21                      0,
22                      0);
23          MONITORING_CONTROLLER.newMonitoringRecord(e);
24      }
25      .
26      .
27      .
28  }
```

Listing 2.3: Extracto de la clase `ImageHandler.java` instrumentalizada con Kieker

Performance Model Extractor (PMX), es una herramienta que automatiza la extracción de modelos de rendmiento a partir de mediciones. PMX utiliza como entrada las bitácoras basadas en Kieker y es capaz de crear modelos basados en *Palladio Component Model* a partir de estas.

Los aspectos relacionados con la estrategia de cómo se obtuvo un modelo de rendimiento a partir de las bitácoras de Kieker y PMX se dan a conocer en la Sección 2.3.

Versión instrumentalizada para AWS X-Ray (IM-XRay)

La principal motivación detrás de esta nueva versión de *Image-Handler* es la de contar con datos del rendimiento de la función Lambda que no pudieron llegar a ser estimados durante el proceso de extracción del modelo utilizando PMX. En la Sección 2.3 se brindan los detalles de lo observado durante el proceso de extracción del modelo con PMX y en dónde encajan los resultados arrojados por esta nueva versión en el modelo.

Para esta versión, Se siguió un enfoque similar al de la Sección 2.2.2 pero en lugar de utilizar la biblioteca de Kieker, se utilizó la biblioteca AWS SDK (*Software Development Kit, SDK*) para crear las trazas y *subsegmentos* de trazas, que son vistas más específicas del comportamiento de la aplicación. Con el uso AWS X-Ray se busca:

1. Obtener datos específicos del rendimiento de la función.
2. Averiguar si las nuevas mediciones logran brindar información acerca de la infraestructura AWS Lambda y su impacto en la ejecución de *Image-Handler*.
3. Exportar los datos de rendimiento a algún formato conocido para su manipulación.

Se modificó la función *Image-Handler* para que puede generar trazas de AWS X-Ray en los mismos puntos en el código en los que se agregó la instrumentalización para Kieker. Un extracto de este código se muestra en el listado 2.4. Cada invocación a las operaciones de procesamiento de la entrada, redimensionamiento y entrega de la respuesta se realizan utilizando el método `AWSRay.createSubsegment`.

```java
public class ImageHandlerXRay implements RequestStreamHandler {
    .
    .
    .
    @Override
    public void handleRequest(InputStream inputStream, OutputStream
    outputStream, Context context) throws IOException {
        ImageRequest imageRequest = AWSXRay.createSubsegment("input
    event", new Function<Subsegment, ImageRequest>() {
            @Override
            public ImageRequest apply(Subsegment subsegment) {
                return inputEventParser().processInputEvent(inputStream
    );
            }
        });

        InputStream imageResized = AWSXRay.createSubsegment("resize
    event", new Function<Subsegment, InputStream>() {
            @Override
            public InputStream apply(Subsegment subsegment) {
                return imageService().getImageFrom(imageRequest);
            }
        });

        AWSXRay.createSubsegment("write response", () -> {
            imageHandlerResponseWriter().writeResponse(imageResized,
    outputStream, imageRequest);
        });
    }
    .
    .
    .
}
```

Listing 2.4: Extracto de la clase `ImageHandler.java` instrumentalizada con AWS X-Ray

En la figura 2.5 se muestran los principales involucrados en la generación de trazas de rendimiento en AWS X-Ray. A la función *Image-Handler* se le agrega la biblioteca de AWS X-Ray para crear las trazas. Estas trazas se envían al servicio AWS X-Ray que es el recolecta estas trazas y con base en ellas se pueden obtener mapas de la interacción de los servicios que componen la función y desgloces de los subsegmentos que componen la traza.

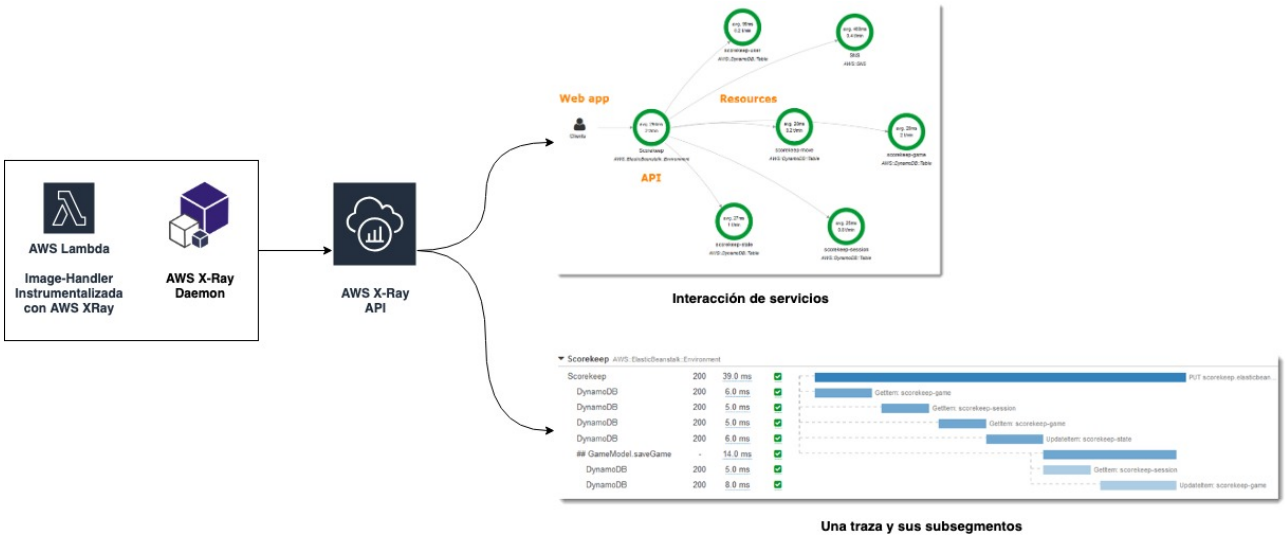

Figura 2.5: *Image-Handler* publicando eventos de rendimiento al servicio AWS X-Ray

2.3. Estrategia de extracción de modelo de rendmiento para *Image-Handler*

Debido a que las funciones Lambda se ejecutan en contenedores que son tanto inaccesibles como efímeros para los diseñadores e implementadores, y, sobre los cuales no se tiene ningún control, estrategias tradicionales en donde se crean bitácoras en la misma computadora en donde se ejecuta la aplicación y se van monitoreando utilizando alguna herramienta especializada o simplemente mediante *Secure Socket Channel*(SSH) deben ser replanteadas. Para este tipo de software, se hace necesario registrar los eventos asociados al comportamiento del rendimiento en una computadora o servicio externo en el cual se tenga control para acceder a los resultados y manipularlos.

Para la extracción del modelo PCM a partir de las bitácoras de Kieker, se llevaron a cabo las siguientes actividades:

1. Creación de la versión Image-PK (Sección 2.2.2): Versión de *Image-Handler*

con las bibliotecas de Kieker para generar bitácoras del rendimiento de la función.

2. Provisionar una nueva máquina virtual en AWS, en la cual se va a:
   - Ejecutar una cola JMS.
   - Ejecutar una aplicación consumidora de mensajes de la cola JMS.
   - Almacenar la bitácora de registros de rendimiento de la función Lambda.
3. Configurar la biblioteca de Kieker para indicar que la publicación de los registros de rendimiento de la función se hagan a través de la cola JMS en la máquina virtual del punto #2.
4. Creación de una aplicación consumidora de mensajes para que una vez que arriben los mensajes a la cola JMS, esta procese los mensajes de la cola y los almacene en una bitácora en la máquina virtual creada en el punto #2. La figura 2.6 muestra los involucrados en el proceso de publicación de mediciones de rendimiento del código de *Image-Handler* hacia una bitácora externa.
5. Una vez obtenida una bitácora en formato Kieker, esta se usó como entrada para PMX. PMX inspecciona la bitácora, la procesa y retorna un archivo `.zip` con los archivos correspondientes a una instancia de PCM. Lo anterior se aprecia en la figura 2.7.

2.3.1. Modelo obtenido

A partir de la bitácora proporcionada como entrada, PMX logró identificar 6 componentes principales:

- `ImageHandlerKieker`: El punto de entrada de la función.
- `ImageRequestParser`: Encargado de tomar la solicitud de redimensionamiento, analizarla y convertirla en un objeto que pueda ser utilizado por el resto de componentes.
- `S3ImageService`: Contiene la lógica de:
  1. Cómo obtener una imagen y
  2. Cómo aplicar la redimiensión sobre la misma.
- `S3Dao`: El componente que sabe cómo obtener una imagen el servicio AWS S3
- `AmazonS3Client`: Contiene las operaciones de comunicación de bajo nivel con el servicio AWS.
- `HandlerResponseWriter`: Convierte la imagen redimensionada a una representación en `Base 64` y prepara la respuesta de la función.

Cada uno de ellos expone su funcionalidad por medio de una interfaz. En el modelado y simulación basado en componentes, los componentes se conciben como piezas intercambiables los cuales exponen sus operaciones por medio de interfaces y delegan los detalles de implementación a componentes concretos (Como por ejemplo *BasicComponents*).

Durante las pruebas realizadas al modelo generado por PMX, se percató que si bien el modelo representaba muy bien la intención detrás de los componentes del código fuente, no era detallado en las estimaciones del uso de cada componente. En PCM, a cada componente se le puede especificar su flujo de acciones, estimaciones de rendimiento e invocaciones a otros componentes, por medio de *Service Effect Specifications* (SEFF).

En principio, las estimaciones incluidas en los SEFFs generados por PMX no lograron ser de utilidad para obtener predicciones representativas de lo observado en las ejecuciones de la función Lambda: mientras que en las invocaciones a *Image-Handler* se entregaban tiempos de respuesta distintos, las simulaciones sobre el modelo entregan siempre un dato fijo. Las estimaciones de rendimiento de los SEFFs se basaban en tiempos de cómputo constantes lo que hacía que el motor de simulaciones generara una predicción del tiempo de respuesta que era el mismo para todos los casos.

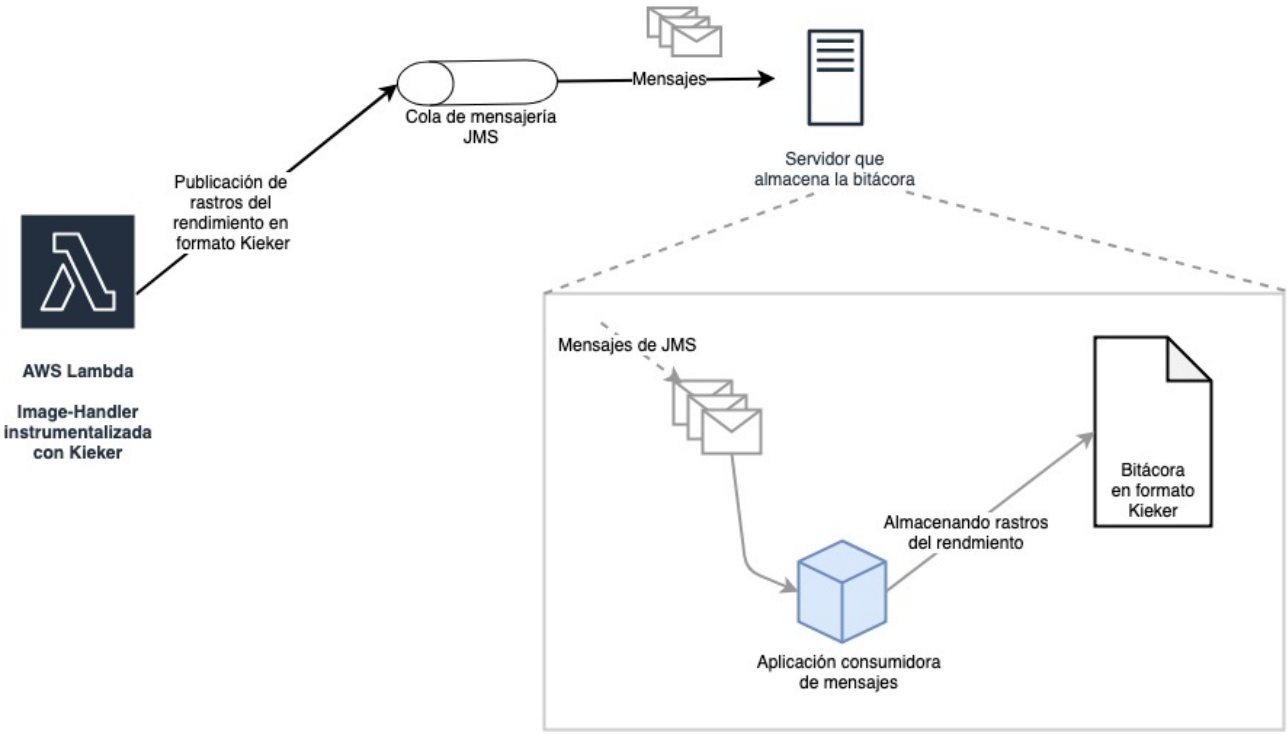

Figura 2.6: Publicando mediciones del rendimiento de la función Lambda.

Aporte de la versión IM-XRay a las simulaciones

Una actividad recurrente durante el modelado y simulación de arquitecturas de software, es la del afinamiento del modelo. Esta tarea es necesaria para

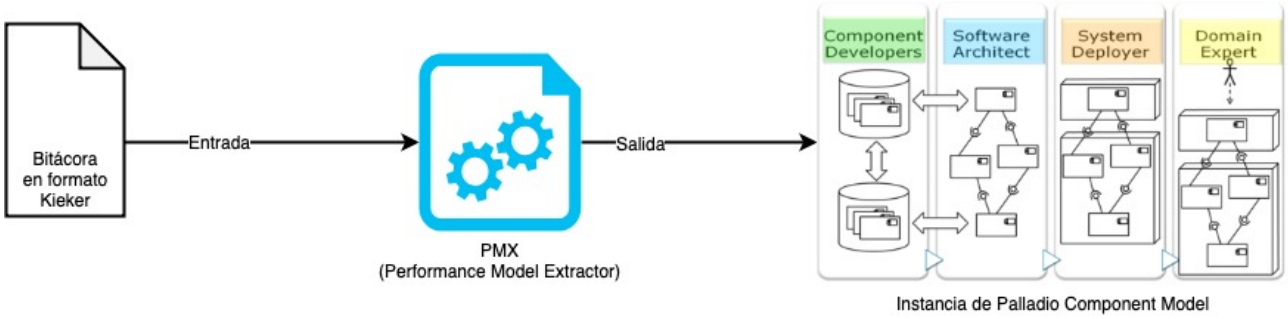

Figura 2.7: Convirtiendo una bitácora de Kieker a una instancia de PCM por medio de PMX.

que el modelo en cual se está trabajando pueda llegar a convertirse una representación cercana del comportamiento de un software en escenarios reales.

Durante el trabajo con el modelo PCM obtenido en la Sección 2.2.2 se notó que las estimaciones hechas por PMX no correspondían al comportamiento observado y que el esfuerzo necesario para obtener estas estimaciones a partir de las bitácoras de Kieker podría llegar a ser grande, principalmente porque el formato de las bitácoras de Kieker incluye muchos más datos que solo las estimaciones de rendimiento y porque se empezó a tener la sensación de que estos datos de alguna u otra forma no estaban "contando toda la historia" de lo que estaba pasando con la función Lambda. Por ejemplo, las datos de rendimiento de Kieker no podían explicar tiempos de retraso que se observaban al inicio de la ejecución de la función Lambda y, sin esos datos, se podía llegar a generar un modelo poco preciso.

Por esta razón se comenzó a explorar herramientas alternativas para obtener mediciones del rendimiento de la función Lambda y se elegió Amazon X-Ray[8] para este fin. AWS XRay ayuda a los desarrolladores a analizar y depurar aplicaciones en producción distribuidas, tal y como lo son las basadas en arquitecturas de microservicios. Con AWS X-Ray se puede ver cómo es que la aplicación

[8] `https://aws.amazon.com/xray/`

y sus servicios asociados se están ejecutando para identificar y resolver problemas de rendimiento y errores en general.

AWS X-Ray recolecta datos de las solicitudes que hacen a cada uno de los servicios de la aplicación y los agrupa en unas unidades llamadas trazas(*traces*). Luego, utilizando estas trazas, es posible ver mapas de la interacción de los servicios, latencias y metadatos para analizar el comportamiento o identificar problemas.

Las trazas recolectadas en AWS X-Ray fueron de gran utilidad para refinar las estimaciones de rendimiento que cada uno de los componentes identificados realizaba. En el experimento #1, en la Sección 2.4.1, se tomaron los datos de rendimiento de cada componente y se realizaron sobre los mismos análisis de frencuencias con el fin de conocer cómo se distribuían estos datos con respecto a sus probabilidades. **Revisar párrafo con ITZ**. Las probabilidades fueron introducidas en los *SEFFs* de cada componente para la ejecución de simulaciones.

2.4. Diseño Experimental

En esta sección se detallan los experimentos realizados para:

- Validar si el modelo y la simulaciones sobre el mismo, logran caracterizar el comportamiento de la función *Image Handler* en distintos escenarios.
- Estudiar el comportamiento de la función Lambda cuando es invocada con cargas de trabajo y
- Comparar los resultados de las invocaciones de la función Lambda con los de la herramienta SAM CLI.

2.4.1. Utilizando *Image-Handler* para redimensionar imágenes de distintos tamaños

Este es el caso que se menciona en la Sección 2.1.1 y se muestra en la figura 2.3. Se realizaron invocaciones a la función Lambda con tres grupos de imágenes:

1. Imágenes de tamaño menor o igual a 500Kb.

2. Imágenes de tamaño mayor a 500Kb y menor a 1Mb.

3. Imágenes de tamaño mayor a 1Mb y menor a 2Mb.

En este experimento, el objetivo es comprobar por medio de mediciones directas y de simulaciones en un modelo, cómo los distintos tamaños de las imágenes influyen en el tiempo de respuesta de la función.

Intuitivamente, se espera que, cuando se hagan solicitudes de redimensionamiento de imágenes de mayor tamaño tomen mayor tiempo en ser procesadas y que lo contrario suceda con las imágenes de menos tamaño. Los resultados obtenidos brindan una referencia inicial para saber cómo es que los componentes de software asociados al redimensionamiento trabajan y qué posibles mejoras podrían realizarse.

Las cargas de trabajo para este experimento son de tipo *cerrada*, lo que quiere decir que una solicitud se ejecuta solamente hasta que la anterior se termina. Esto va orientado a tener mejor trazabilidad de lo que ocurre con la función.

Invocaciones con imágenes menores a 500Kb

Para la realización de este experimento se contó con la siguiente configuración base:

- *Sujeto de prueba:* La función Lambda IM-Simple.
- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño menor a 500Kb alojadas en Amazon S3.
- *Carga de trabajo:* 1000 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-Simple.
- *Herramientas de medición:* Amazon Cloudwatch.

Configuración para la obtención de datos de rendimiento

- *Sujeto de prueba:* Las funciones Lambda IM-KP y IM-XRay
- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño menor a 500Kb alojadas en Amazon S3.
- *Carga de trabajo:* 1000 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-PK y IM-XRay.
- *Herramientas de medición:* Kieker, PMX y Amazon X-Ray.

Para la realización de este experimento, se obtuvieron 1000 imágenes aleatorias de tamaño menor a 500Kb del servicio *Lorem Picsum*[9]. Se creó un *script* en `Bash` para acceder a la interfaz de programación (API) proporcionada por *Lorem Picsum* para descargar de forma aleatoria 1000 imágenes cuyo tamaño era

[9] `https://picsum.photos`

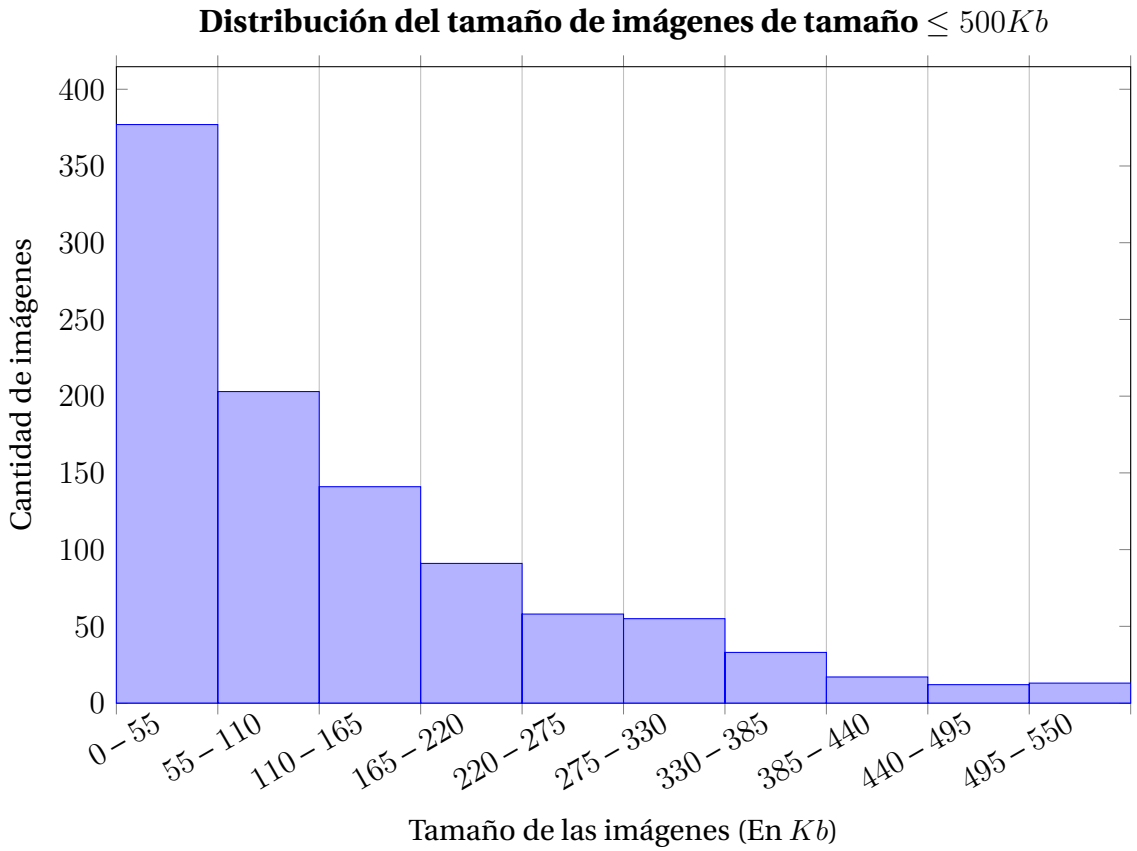

Figura 2.8: Distribución del tamaño de imágenes $\leq 500Kb$

menor a los 500Kb. La distribución del tamaño de las 1000 imágenes, en Kb, se aprecia en la figura 2.8. El mismo grupo de imágenes se utilizó para realizar solicitudes de redimensionamiento sobre IM-Simple, IM-KP y IM-XRay.

Medición Base: 1000 invocaciones de redimensionamiento de imágenes en IM-Simple. Se creó un *script* en `Bash` para ejecutar 1000 invocaciones de redimensionamiento en la función IM-Simple en las imágenes de tamaño menor a 500Kb. El *script* selecciona una imagen de forma aleatoria y luego ejecuta la solicitud de redimensionamiento utilizando dimensiones de ancho y alto de uso común para imágenes en miniatura (*thumbnails*) **AGREGAR APARTADO SOBRE LA ELECCION LOS THUMBNAILS**.

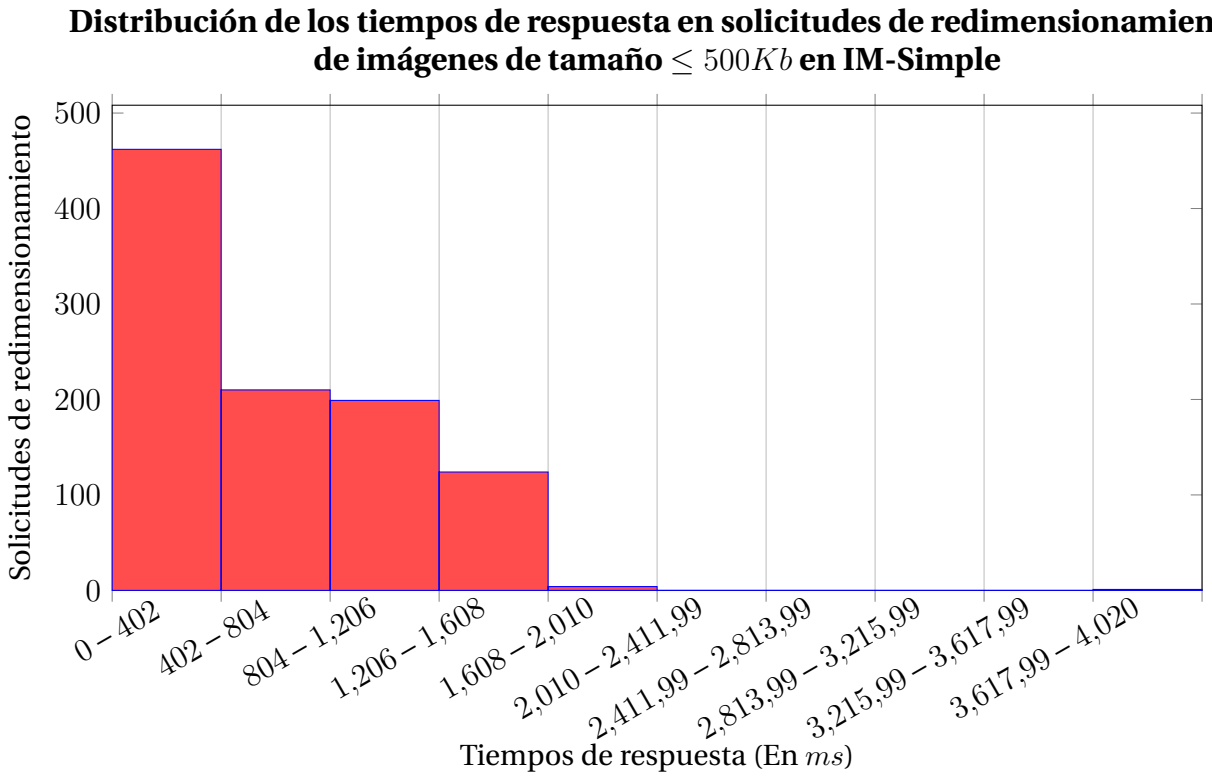

Figura 2.9: Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $\leq 500Kb$ en IM-Simple

En la figura 2.9 se muestra la distribución de los tiempos de respuesta en las solicitudes de redimensionamiento en las imágenes de tamaño menor a 500Kb. Con excepción de la primera invocación, la cual tuvo una duración de 4 segundos, más del 97,5 % de las invocaciones no superó los 1,6 segundos.

Mediciones para obtención de modelo de rendimiento: 1000 invocaciones de redimensionamiento de imágenes en IM-PK y IM-XRay. Se utilizó el mismo *script* en `Bash` y la misma configuración para generar invocaciones a la función Lambda descrita en la sección anterior.

En primera instancia se ejecutaron 1000 invocaciones a IM-PK para generar una bitácora de Kieker y a partir de la misma extraer un modelo de rendimiento PCM usando PMX. Tal y como se señala en la Sección 2.3.1, las estimaciones

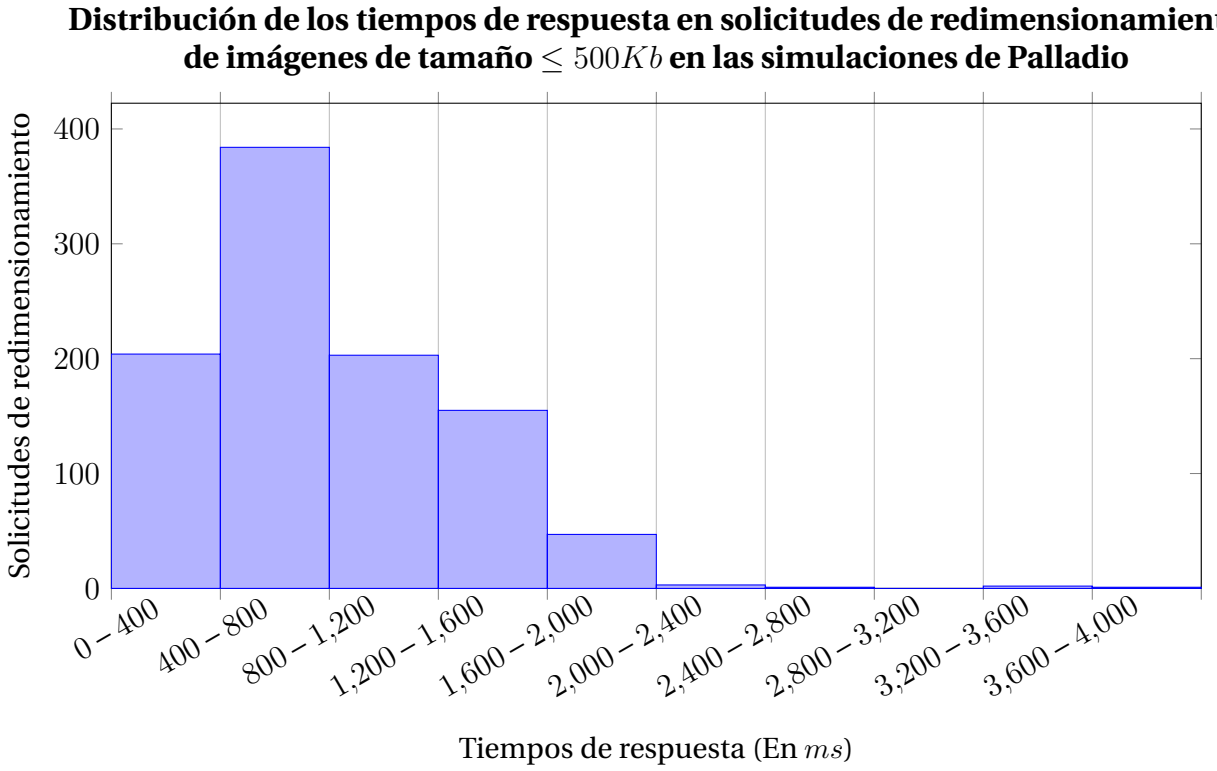

Figura 2.10: Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $\leq 500Kb$ en las simulaciones de *Palladio Workbench*

hechas por PMX sobre el rendimiento de los componentes del modelo se basaban en valores constantes. Fue por esta razón que, para contar con una versión alternativa de *Image Handler* que pudiera brindar otro nivel de detalle en las métricas de rendimiento, se introdujo IM-XRay.

Al igual que en caso anterior, se ejecutaron 1000 invocaciones a IM-XRay, y por medio de un *script* en `Bash`, se obtuvieron las trazas correspondientes a las 1000 invocaciones. Los nuevos datos fueron exportados a formato `.csv` e interpretados con el lenguaje `R`. En `R`, se calcularon distribuciones de frecuencia de la probabilidad en la que un componente lograba procesar una porción de la carga de trabajo total. Estos datos fueron incluidos en los *SEEFs* de cada componente del modelo. Por último se ejecutó una simulación en *Palladio Workbench* con los siguientes parámetros:

Hasta 500Kb			
Solicitud de redimensionamiento	IM-Simple	PCM	Diferencia
Tiempo promedio	583.842ms	793.808ms	209.965ms
Desviación estándar	460.659ms	465.441ms	4.782ms
Varianza	212206.961	216635	–
Mediana	466.715ms	680.482ms	. –
Coeficiente de variación	0.987	0.683	–

Tabla 2.1: Resumen de datos estadísticos

- Generación de 1000 mediciones.
- Carga de trabajo: *cerrada*. Se ejecuta una solicitud sobre el modelo hasta que la anterior termina.

En la figura 2.10, se muestra la distribución de los tiempos de respuesta en las solicitudes de redimensionamiento en las simulaciones de *Palladio Workbench* para imágenes de tamaño $\leq 500Kb$. En los resultados de las simulaciones, el 95 % de las invocaciones no superó los 1,6 segundos en procesar la solicitud de redimensionamiento.

En este punto, se cuenta con 1000 mediciones hechas sobre IM-Simple y un modelo al que se le simularon 1000 invocaciones. En la figura 2.11 se comparan los tiempos de respuesta obtenidos en IM-Simple y los de las simulaciones, y, en el Cuadro 2.1, un resumen de los datos estadísticos de los tiempos de respuesta en ambos sujetos de prueba.

Análisis de resultados

La Figura 2.11 muestra un panorama alentador. Las ejecuciones de las simulaciones en PCM presentan tiempos de respuesta muy similares a los que entrega IM-Simple. Hay una diferencia de 209,965ms en el tiempo promedio de

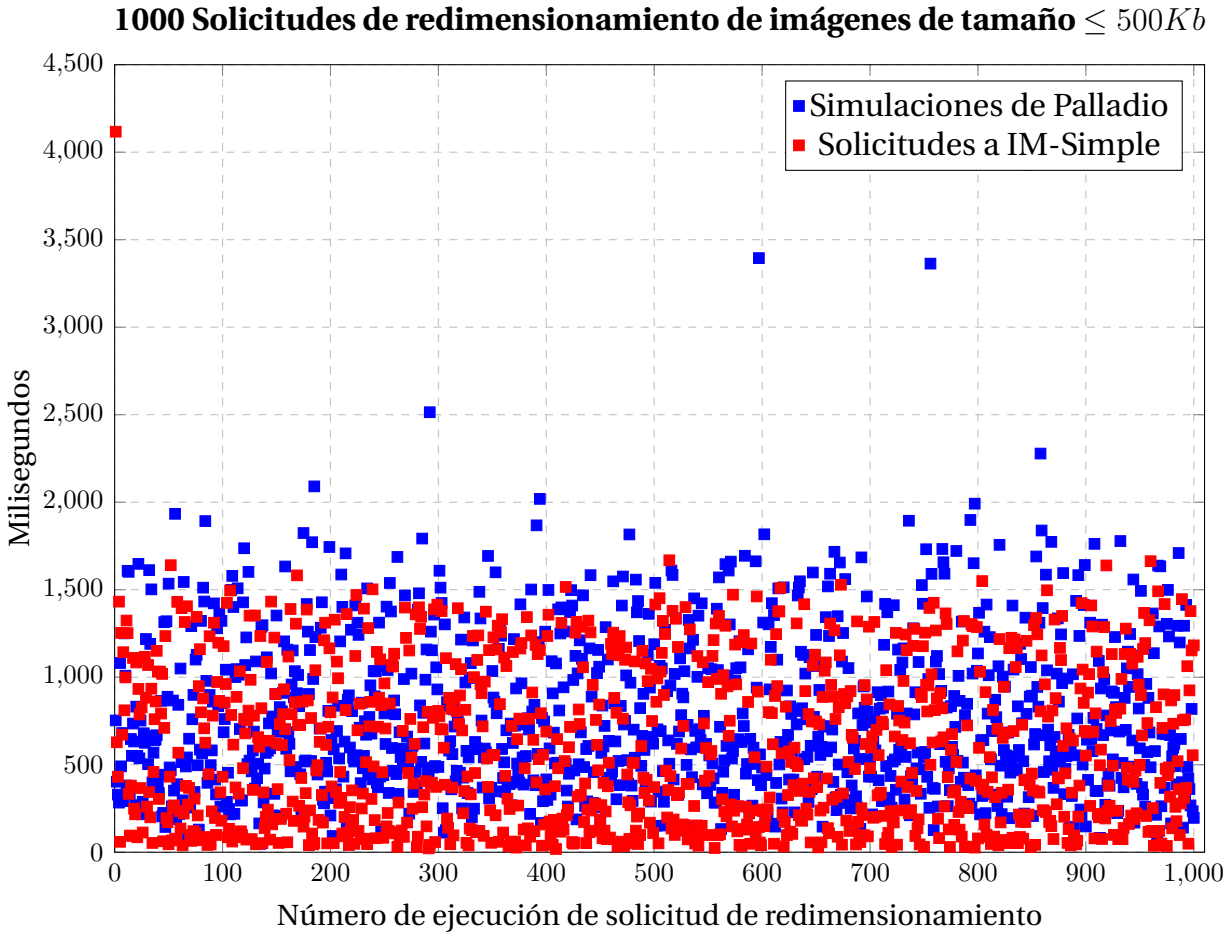

Figura 2.11: IM-Simple *vs* simulaciones en PCM: 1000 solicitudes de redimensionamiento de imágenes de tamaño $\leq 500Kb$.

los tiempos de respuesta de las simulaciones en PCM con respecto a los tiempos de IM-Simple. Preliminarmente, se valora que, debido a que la versión IM-XRay tiene activado el servicio de monitoreo AWS X-Ray y que fue esta versión de *Image Handler* utilizada como referencia para generar los tiempos procesamiento estimados para cada componente del modelo, instrumentalizar la función Lambda con el servicio de monitoreo AWS X-Ray genera un *overhead* en el procesamiento de la función. Cabe mencionar que, la estrategia de monitoreo utilizada fue muy agresiva debido a que para obtener nuevas métricas para los componentes del modelo PCM, se habilitaron muchos puntos de monitoreo dentro del código fuente. Además, se configuró la función Lambda para que

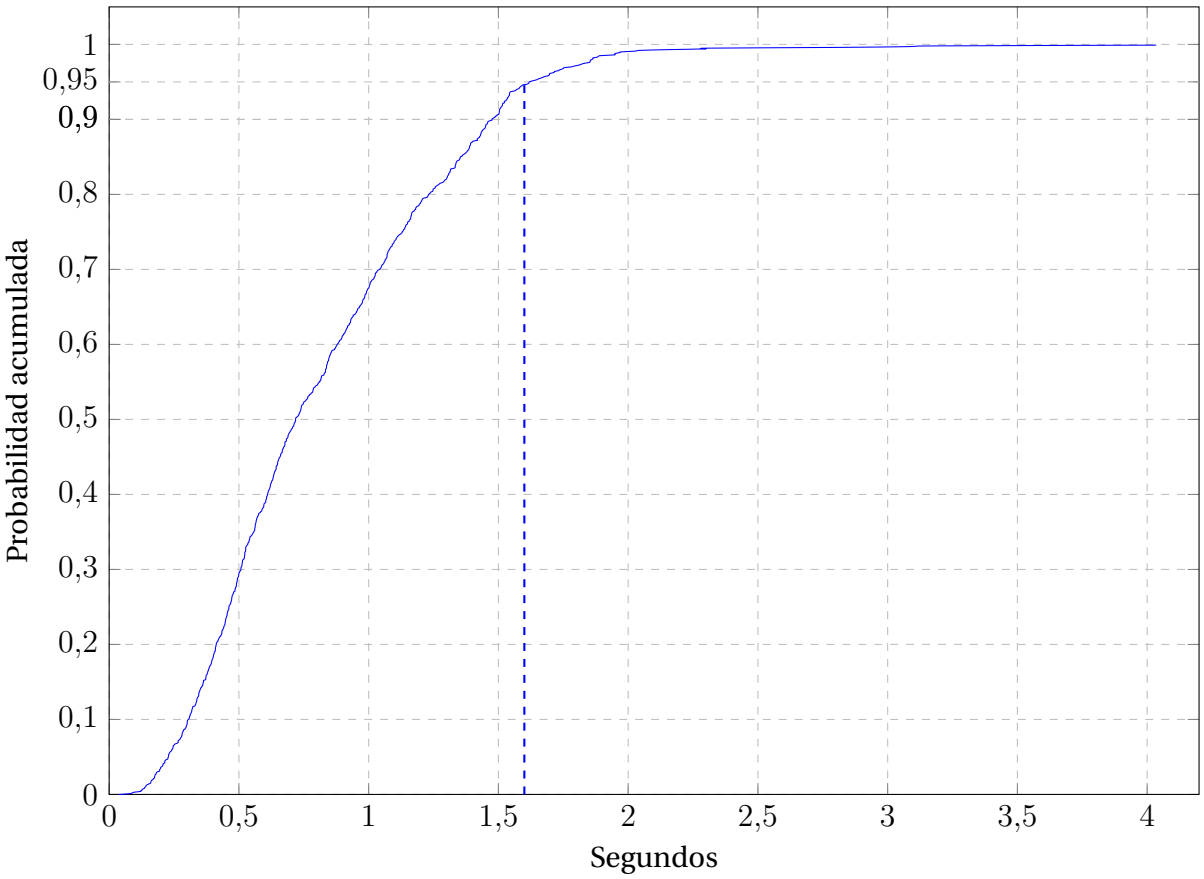

Figura 2.12: Probabilidad acumulada en solicitudes de redimensionamiento en imágenes de tamaño $\leq 500Kb$ en *Palladio Workbench*

monitoreara el 100 % de las invocaciones a solicitudes de redimensionamiento. Esta configuración de monitoreo no es la habitual que se utiliza para las funciones Lambda en producción, pero, para el caso de este estudio se necesitó contar con mayores niveles de detalle en las mediciones por lo que fue necesario sacar el mayor provecho al monitoreo. Entre menor sea utilizado las opciones de monitoreo de AWS X-Ray en la función Lambda, menor será el *overhead* experimentado, tal y como pasa con IM-Simple en donde no se experiementa *overhead* por concepto de monitoreo.

Los resultados muestran desviaciones estándar de 460,569ms y 465.445ms para IM-Simple y las simulaciones de PCM respectivamente. La desviación es-

tándar de las simulaciones de PCM es solamente 4,782ms mayor que la de IM-Simple, lo que sugiere que la agrupación de los datos con respecto a su media aritmética serán muy semejantes.

Los coeficientes de variación para IM-Simple y las simulaciones de PCM fue de 0.987 y 0.683 respectivamente. Estos resultados, apuntan a una mayor heterogeneidad entre los tiempos de respuesta y a que el tiempo promedio de procesamiento no se considere representativo para este conjunto de datos. Esta variabilidad viene dada por las diferencias de los tamaños de las imágenes utilizadas para este experimento, como se muestra en la Figura 2.8: imágenes de tamaño $\leq 500Kb$, en donde existen diferencias de hasta 500x, por lo que, por ejemplo, el tiempo de procesamiento de una imagen de tamaño de 5kb será más rápido que una de tamaño de 490Kb.

Por último, de acuerdo con los resultados de las simulaciones, existe un 95 % de probabilidad de que el tiempo de procesamiento de una solicitud de redimensionamiento de una imagen de tamaño $\leq 500Kb$ tome 1,6 segundos o menos (Figura 2.12). En IM-Simple, se obtuvo un 97,5 % de probabilidad para el mismo caso (Figura 2.9). Para este caso en particular y, debido a la variabilidad de los tiempos de respuesta, se considera que el uso de esta probabilidad acumulada es más representativa a la hora de describir el comportamiento de la función Lambda.

Invocaciones con imágenes mayores a 500Kb y menores o igual a 1Mb

Para la realización de este experimento se contó con la siguiente configuración base:

- *Sujeto de prueba:* La función Lambda IM-Simple.
- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb alojadas en Amazon S3.
- *Carga de trabajo:* 1000 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-Simple.
- *Herramientas de medición:* Amazon Cloudwatch.

Configuración para la obtención de datos de rendimiento

- *Sujeto de prueba:* La función IM-XRay
- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb alojadas en Amazon S3.
- *Carga de trabajo:* 1000 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-XRay.
- *Herramientas de medición:* Amazon X-Ray.

Para este experimento no realizó ningún trabajo sobre la versión IM-KP porque no se necesita extraer ningún modelo. El modelo fue extraído y generado en el experimento anterior y debido a que se usa es el mismo código con los mismos puntos de monitoreo, un eventual proceso de extracción daría como resultado el mismo modelo del experimento anterior. Lo que sí fue necesario hacer, fue calibrar el modelo existente con los resultados de las mediciones a solicitudes de redimensionamiento en imágenes de tamaño mayor a 500Kb y menor o igual 1Mb, utilizando la versión IM-XRay.

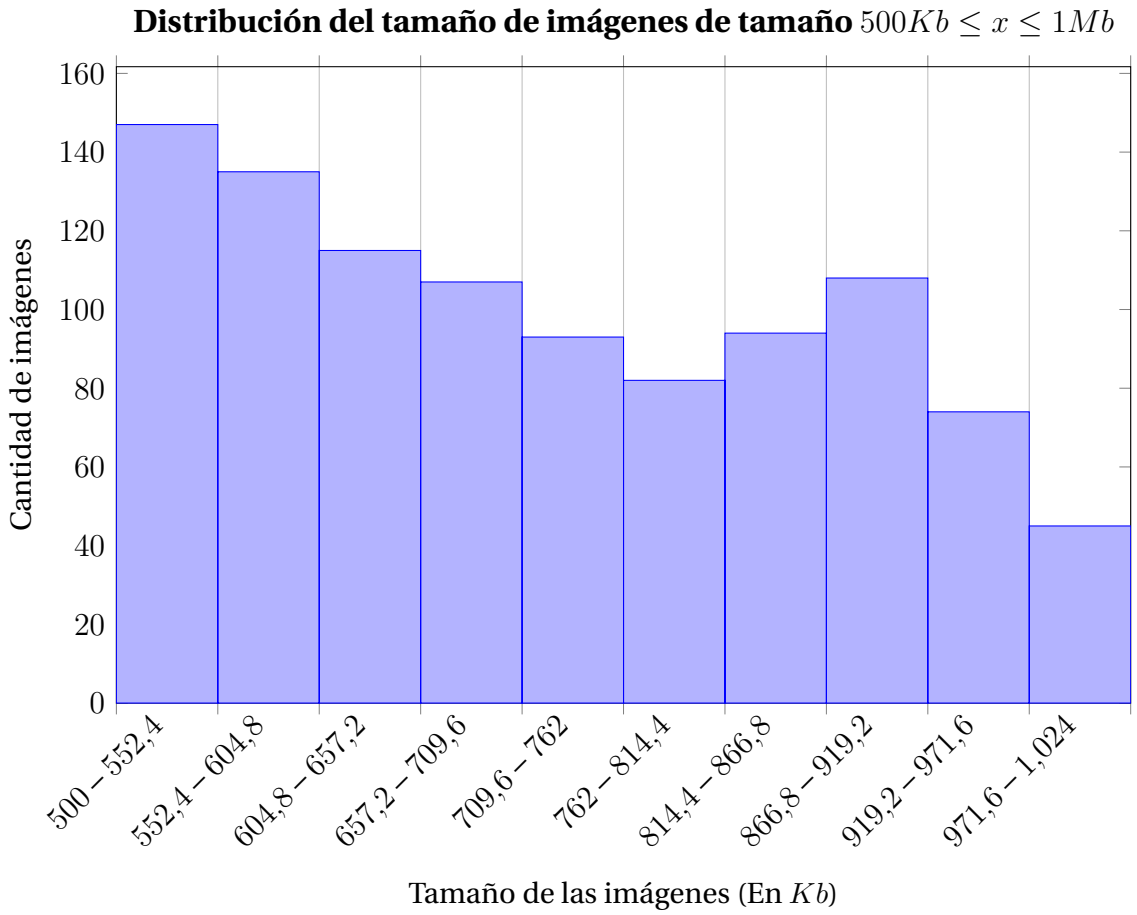

Figura 2.13: Distribución del tamaño de imágenes $500Kb \leq x \leq 1Mb$

Para la realización de este experimento, se obtuvieron 1000 imágenes aleatorias de tamaño mayor a 500Kb y menor o igual a 1Mb del servicio *Lorem Picsum*. Se ejecutó el *script* en `Bash` del experimento anterior para acceder a la API de *Lorem Picsum* para descargar de forma aleatoria 1000 imágenes cuyo tamaño fuera mayor a los 500Kb y menor o igual a 1Mb. La distribución del tamaño de las 1000 imágenes, en Kb, se aprecia en la figura 2.13. El mismo grupo de imágenes se utilizó para realizar solicitudes de redimensionamiento sobre IM-Simple y IM-XRay.

Medición Base: 1000 invocaciones de redimensionamiento de imágenes en IM-Simple. Se ejecutó el *script* en `Bash` del experimento anterior para ejecu-

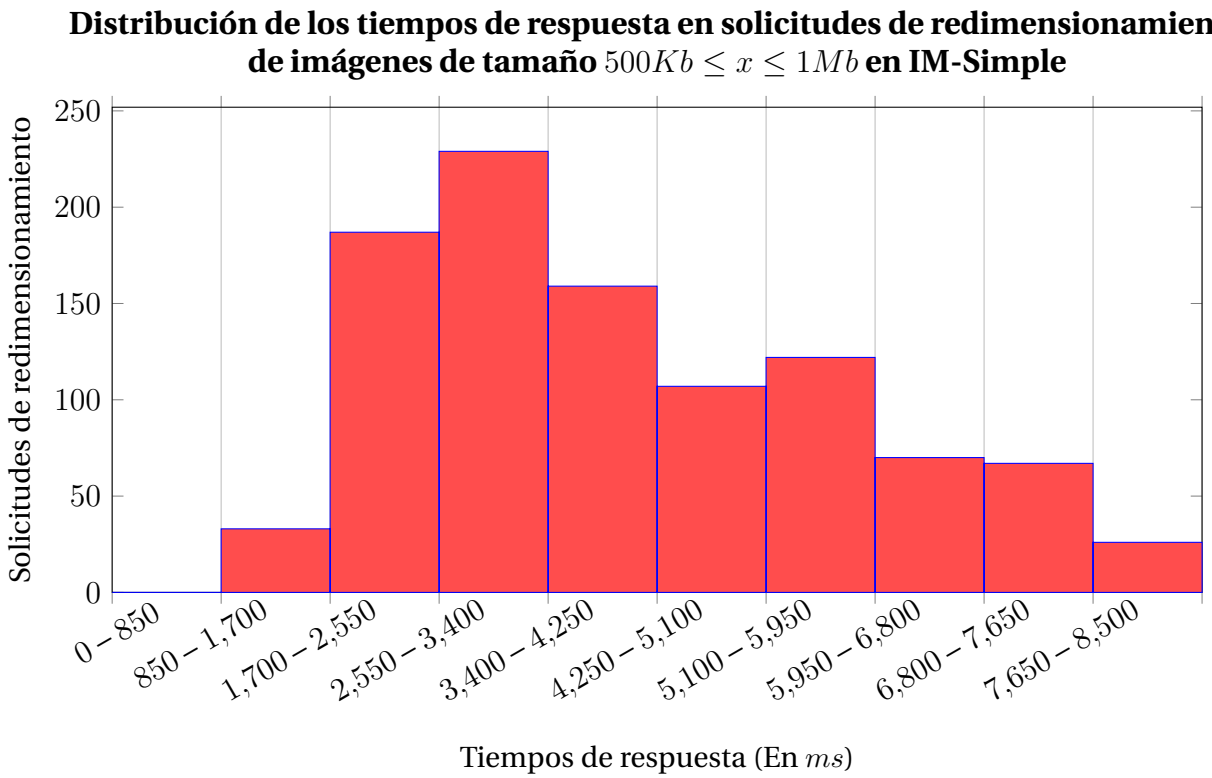

Figura 2.14: Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $500Kb \leq x \leq 1Mb$ en IM-Simple

tar 1000 invocaciones de redimensionamiento en la función IM-Simple en las imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb. El *script* selecciona una imagen de forma aleatoria y luego ejecuta la solicitud de redimensionamiento utilizando dimensiones de ancho y alto de uso común para imágenes en miniatura.

En la figura 2.14 se muestra la distribución de los tiempos de respuesta en las solicitudes de redimensionamiento en las imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb. Más del 98 % de las invocaciones no superó los 8 segundos.

Mediciones para la calibración de modelo de rendimiento: 1000 invocaciones de redimensionamiento de imágenes en IM-XRay. Se utilizó el mismo *script* en `Bash` y la misma configuración para generar invocaciones a la función Lamb-

da descrita en la sección anterior.

Se ejecutaron 1000 invocaciones a IM-XRay, y por medio de un *script* en `Bash`, se obtuvieron las trazas correspondientes a las 1000 invocaciones. Los nuevos datos fueron exportados a formato `.csv` e interpretados con el lenguaje `R`. En `R`, se calcularon distribuciones de frecuencia de la probabilidad en la que un componente lograba procesar una porción de la carga de trabajo total. Estos datos fueron incluidos en los *SEEFs* de cada componente del modelo. Por último se ejecutó una simulación en *Palladio Workbench* con los siguientes parámetros:

- Generación de 1000 mediciones.
- Carga de trabajo: *cerrada*. Se ejecuta una solicitud sobre el modelo hasta que la anterior termina.

En la figura 2.15, se muestra la distribución de los tiempos de respuesta en las solicitudes de redimensionamiento en las simulaciones de *Palladio Workbench* para imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb. En los resultados de las simulaciones, el 95 % de las invocaciones no superó los 8 segundos en procesar la solicitud de redimensionamiento.

En este punto, se cuenta con 1000 mediciones hechas sobre IM-Simple y un modelo al que se le simularon 1000 invocaciones. En la figura 2.16 se comparan los tiempos de respuesta obtenidos en IM-Simple y los de las simulaciones, y, en el Cuadro 2.2, un resumen de los datos estadísticos de los tiempos de respuesta en ambos sujetos de prueba.

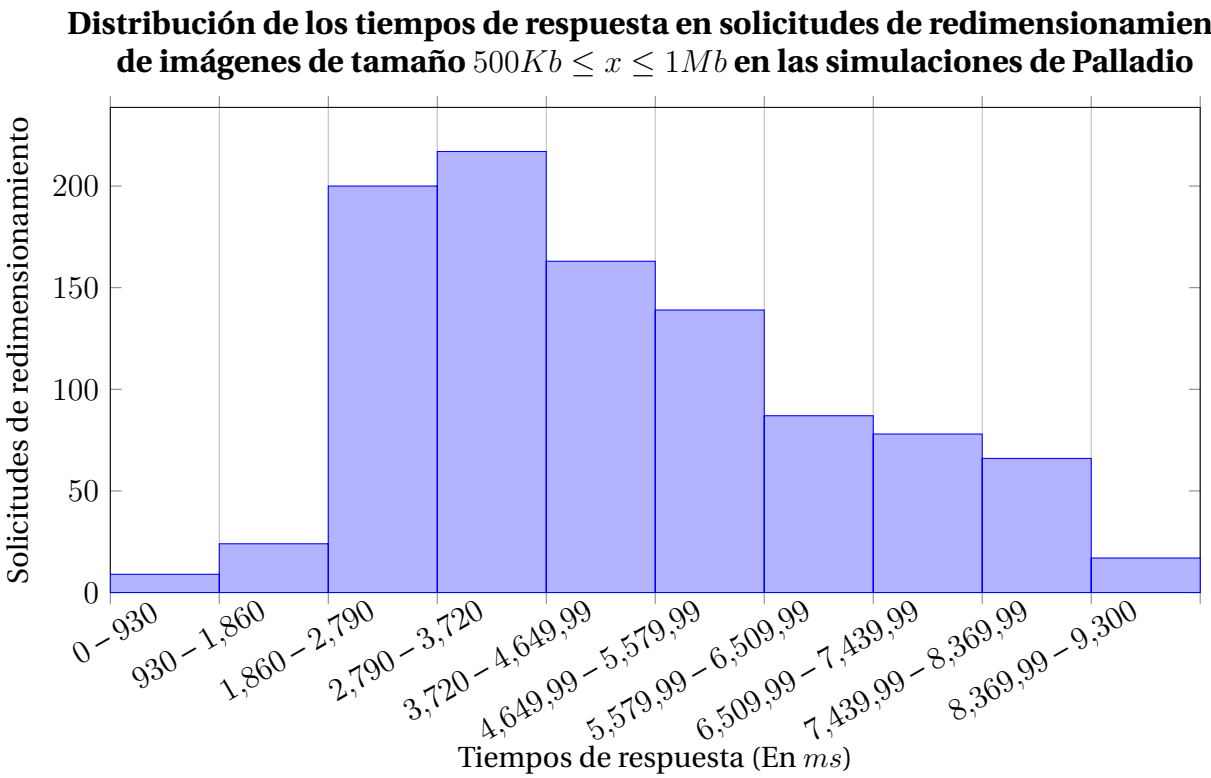

Figura 2.15: Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $500Kb \leq x \leq 1Mb$ en las simulaciones de Palladio

Entre 500Kb a 1Mb			
Solicitud de redimensionamiento	Image Handler	Palladio	Diferencia
Tiempo promedio	4073.600ms	4348.029ms	274.428ms
Desviación estándar	1731.974ms	1844.893ms	112.919ms
Varianza	2999736.844	3403633.84	–
Mediana	3658.825ms	3989.406ms	–
Coeficiente de variación	0.473	0.462	–

Tabla 2.2: Resumen de datos estadísticos

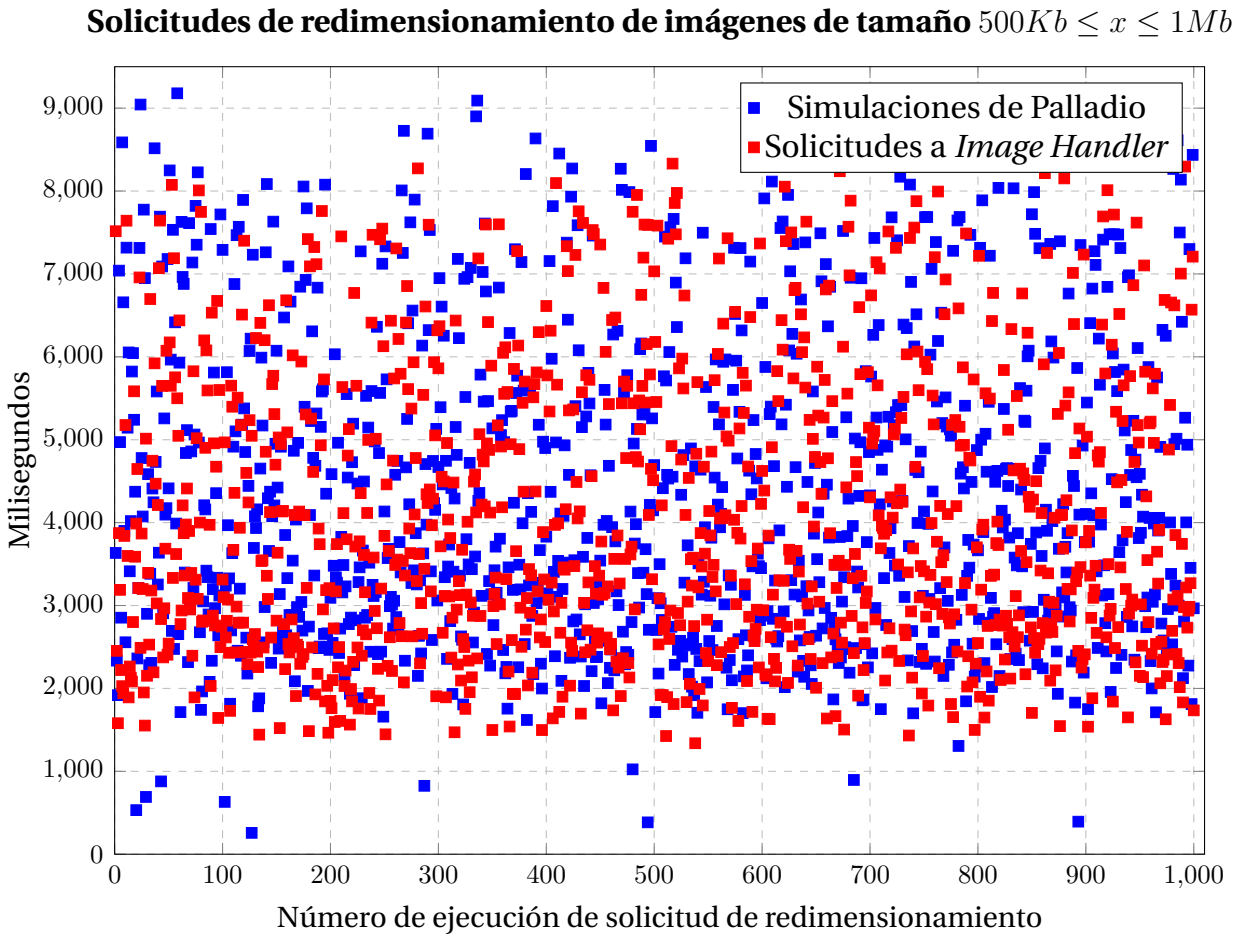

Figura 2.16: Solicitudes de redimensionamiento de imágenes de tamaño $500Kb \leq x \leq 1Mb$

Análisis de resultados

A primera vista, la comparación de los tiempos de respuesta de IM-Simple y las simulaciones en PCM mostradas en la Figura 2.16 muestran gran similitud. Hay una diferencia de 274,428ms en el tiempo promedio de los tiempos de respuesta en las simulaciones PCM con respecto a los tiempos de IM-Simple. En este experimento también se intuye que la estrategia de monitoreo de AWS X-Ray es la resposable de agregar este *overhead* en el procesamiento.

Los resultados muestran desviaciones estándar de 1731.974ms y 1844.893ms para IM-Simple y las simulaciones de PCM respectivamente. La desviación es-

tándar de las simulaciones de PCM es 112.919ms mayor que la de IM-Simple. Esto indica que la agrupación de los datos con respecto a su media aritmética es muy similar entre ambas muestras.

Los coeficientes de variación de ambas muestras bajaron con respecto a los obtenidos en el primer experimento, aún así, los valores de 0,473 para IM-Simple y de 0,462 para las simulaciones de PCM se consideran heterogéneos, no muy similares entre si. Esto hace que el tiempo promedio de una solicitud pueda considerarse no tan representativo a la hora de caracterizar el comportamiento de la función Lambda bajo esta carga de trabajo. La variabilidad en los tiempos de respuesta bajó debido a que en los tamaños de las imágenes utilizadas (Figura 2.13) tienen una diferencia de hasta 2 veces: es decir, en este conjunto, la imagen más grande es el doble de grande que la imagen más pequeña.

Por último, de acuerdo con los resultados de las simulaciones, existe un 95 % de probabilidad de que el tiempo de procesamiento de una solicitud de redimensionamiento de una imagen de tamaño $500Kb \leq x \leq 1Mb$ tome 8 segundos o menos (Figura 2.17). En IM-Simple, se obtuvo un 98 % de probabilidad para el mismo caso (Figura 2.14). Para este caso, como en el experimento anterior, se considera más adecuado el uso de la probabilidad acumulada para describir el comportamiento de la función Lambda.

Invocaciones con imágenes mayores a 1Mb y menores o igual a 2Mb

Para la realización de este experimento se contó con la siguiente configuración base:

- *Sujeto de prueba:* La función Lambda IM-Simple.

Probabilidad acumulada: solicitudes de redimensionamiento en imágenes $500Kb \leq x \leq 1Mb$ en PCM

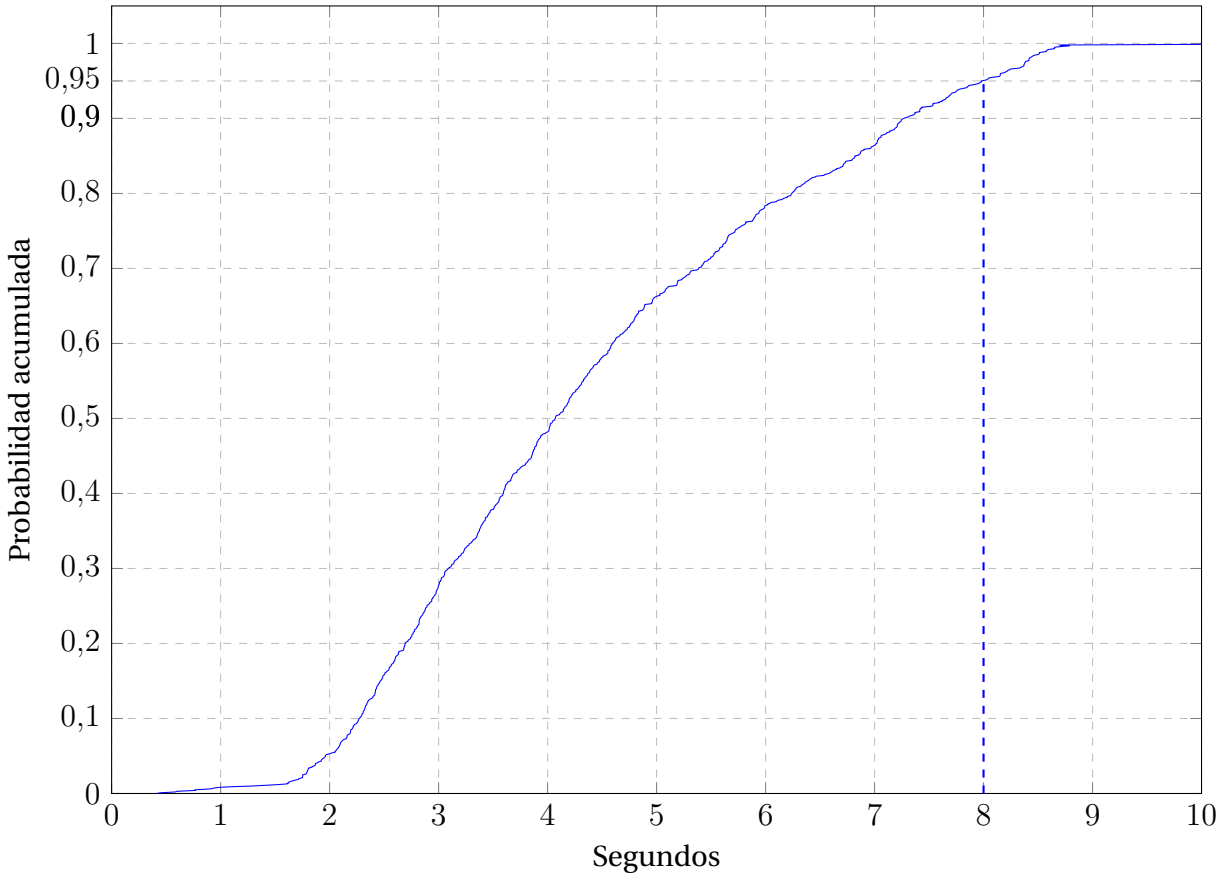

Figura 2.17: Probabilidad acumulada de solicitudes de redimensionamiento en imágenes $500Kb \leq x \leq 1Mb$ en PCM

- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño mayor a 1Mb y menor o igual a 2Mb alojadas en Amazon S3.
- *Carga de trabajo:* 1000 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-Simple.
- *Herramientas de medición:* Amazon Cloudwatch.

Configuración para la obtención de datos de rendimiento

- *Sujeto de prueba:* La función IM-XRay
- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño mayor a 1Mb y menor o igual a 2mb alojadas en Amazon S3.
- *Carga de trabajo:* 1000 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-XRay.
- *Herramientas de medición:* Amazon X-Ray.

Al igual que en el experimento de la sección anterior, no se realizó ningún trabajo sobre la versión IM-KP debido a que no era necesario la extracción de un nuevo modelo, sino de, calibrar el modelo existente con los resultados de las mediciones para generar nuevas simulaciones.

Para la realización de este experimento, se obtuvieron 1000 imágenes aleatorias de tamaño mayor a 1Mb y menor o igual a 2Mb del servicio *Lorem Picsum*. Se ejecutó el *script* en `Bash` del experimento anterior para acceder a la API de *Lorem Picsum* para descargar de forma aleatoria 1000 imágenes cuyo tamaño fuera mayor a 1Mb y menor o igual a 2Mb. La distribución del tamaño de las

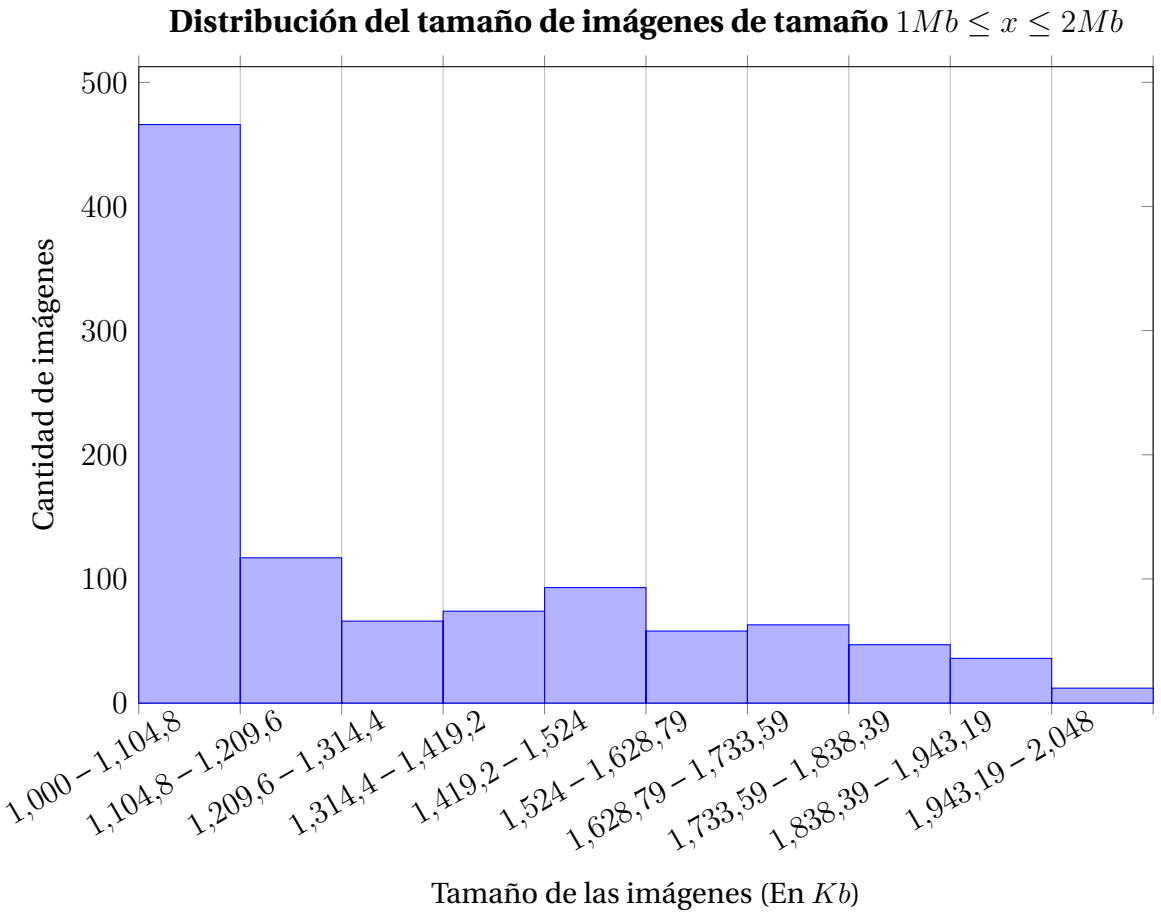

Figura 2.18: Distribución del tamaño de imágenes de tamaño $1Mb \leq x \leq 2Mb$

1000 imágenes, en Kb, se aprecia en la figura 2.18. El mismo grupo de imágenes se utilizó para realizar solicitudes de redimensionamiento sobre IM-Simple y IM-XRay.

Medición Base: 1000 invocaciones de redimensionamiento de imágenes en IM-Simple. Se ejecutó el *script* en `Bash` del experimento anterior para ejecutar 1000 invocaciones de redimensionamiento en la función IM-Simple en las imágenes de tamaño mayor a 1Mb y menor o igual a 2Mb. El *script* selecciona una imagen de forma aleatoria y luego ejecuta la solicitud de redimensionamiento utilizando dimensiones de ancho y alto de uso común para imágenes en miniatura.

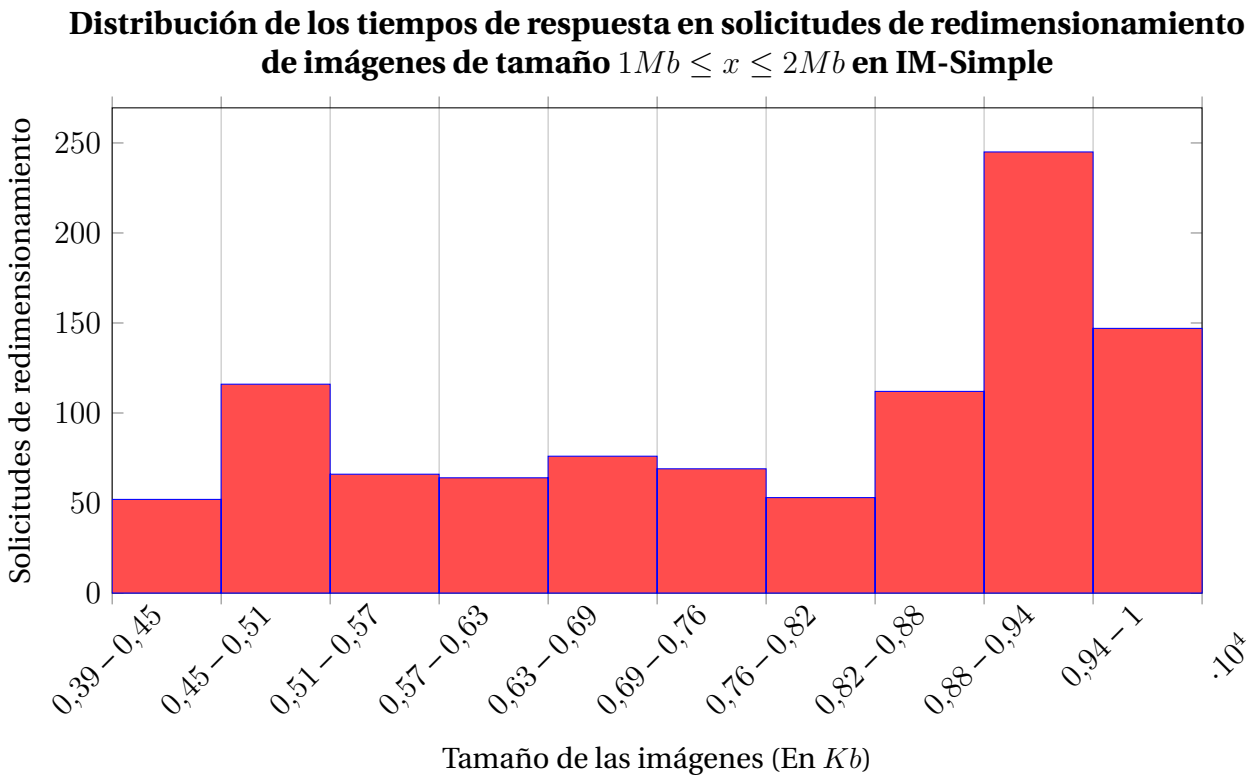

Figura 2.19: Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $1Mb \leq x \leq 2Mb$ en IM-Simple

En la figura 2.19 se muestra la distribución de los tiempos de respuesta en las solicitudes de redimensionamiento en las imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb. Más del 92,5 % de las invocaciones no superó los 9,7 segundos.

Mediciones para la calibración de modelo de rendimiento: 1000 invocaciones de redimensionamiento de imágenes en IM-XRay. Se utilizó el mismo *script* en `Bash` y la misma configuración para generar invocaciones a la función Lambda descrita en la sección anterior.

Se ejecutaron 1000 invocaciones a IM-XRay, y por medio de un *script* en `Bash`, se obtuvieron las trazas correspondientes a las 1000 invocaciones. Los nuevos datos fueron exportados a formato `.csv` e interpretados con el lenguaje `R`. En `R`, se calcularon distribuciones de frecuencia de la probabilidad en la que un componente lograba procesar una porción de la carga de trabajo total. Estos datos fueron incluidos en los *SEEFs* de cada componente del modelo. Por último se ejecutó una simulación en *Palladio Workbench* con los siguientes parámetros:

- Generación de 1000 mediciones.
- Carga de trabajo: *cerrada*. Se ejecuta una solicitud sobre el modelo hasta que la anterior termina.

En la figura 2.20, se muestra la distribución de los tiempos de respuesta en las solicitudes de redimensionamiento en las simulaciones de *Palladio Workbench* para imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb. En los resultados de las simulaciones, el 95 % de las invocaciones no superó los 8 segundos en procesar la solicitud de redimensionamiento.

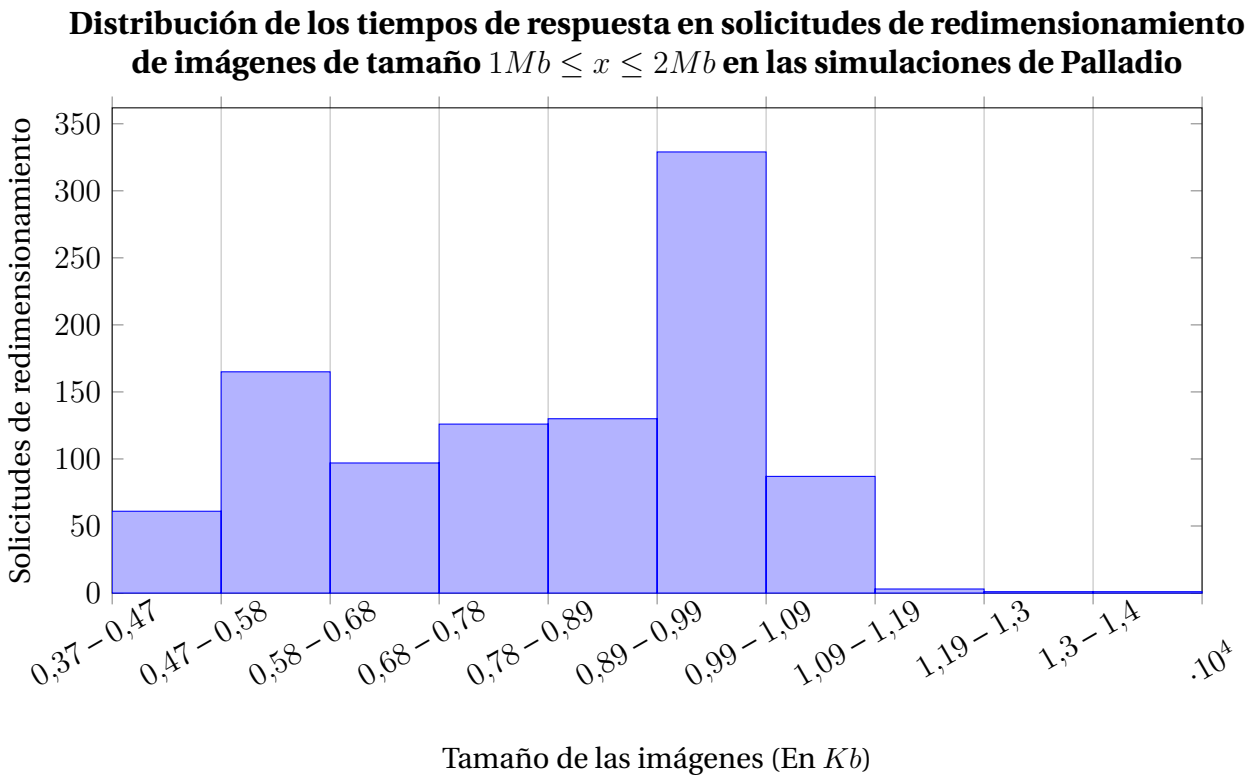

Figura 2.20: Distribución de los tiempos de respuesta en solicitudes de redimensionamiento de imágenes de tamaño $1Mb \leq x \leq 2Mb$ en las simulaciones de Palladio

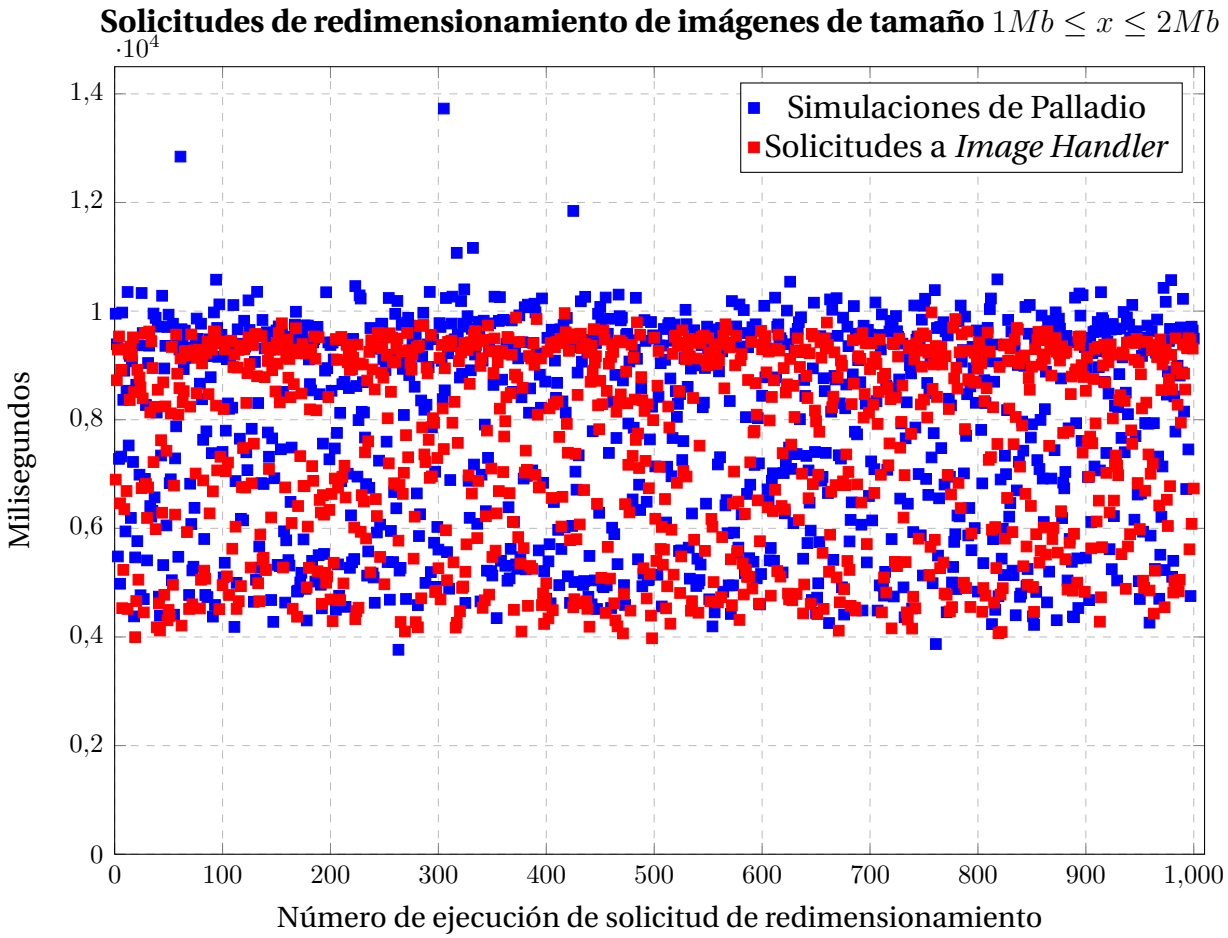

Figura 2.21: Solicitudes de redimensionamiento de imágenes de tamaño $1Mb \leq x \leq 2Mb$

En este punto, se cuenta con 1000 mediciones hechas sobre IM-Simple y un modelo al que se le simularon 1000 invocaciones. En la figura 2.21 se comparan los tiempos de respuesta obtenidos en IM-Simple y los de las simulaciones, y, en el Cuadro 2.3, un resumen de los datos estadísticos de los tiempos de respuesta en ambos sujetos de prueba.

Análisis de resultados

Bajo esta carga de trabajo es en donde se pueden apreciar mayor similitud en los tiempos de procesamiento de las solicitudes de redimensionamiento

Entre 1Mb y 2Mb			
Tiempo promedio	7539.139ms	7796.913ms	257.773ms
Desviación estándar	1816.152ms	1914.258ms	98.106ms
Varianza	3298410.017	3664385	–
Mediana	8200.875ms	8310.293ms	–
Coeficiente de variación	0.221	0.230	–

Tabla 2.3: Resumen de datos estadísticos

entregados por IM-Simple y las simulaciones de PCM. Hay una diferencia de 257,773ms en el tiempo promedio de los tiempos de respuesta en las simulaciones PCM con respecto a los tiempos de IM-Simple. Al igual que en los experimientos anteriores, se considera que la influencia del servicio de monitoreo de AWS X-Ray es el responsable de causar la diferencia en el tiempo de procesamiento.

Los resultados muestran desviaciones estándar de 1816,152ms y 1914.258ms para IM-Simple y las simulaciones de PCM respectivamente. La desviación estándar de las simulaciones de PCM es 98.106ms mayor que la de IM-Simple. Esto indica que la agrupación de los datos con respecto a su media aritmética es muy similar entre ambas muestras.

Los coeficientes de variación de son de 0,221 para IM-Simple y de 0,230 para las simulaciones de PCM. Estos resultados son los menores de los tres experimentos y sugieren que los datos son homogéneos entre sí y que, para esta muestra se pueda considerar representativo utilizar el tiempo promedio de respuesta para caracterizar el comportamiento de la función Lambda.

se consideran heterogéneos, no muy similares entre si. Esto hace que el tiempo promedio de una solicitud pueda considerarse no tan representativo a la hora de caracterizar el comportamiento de la función Lambda bajo esta carga de trabajo. La variabilidad en los tiempos de respuesta bajó debido a que en los

tamaños de las imágenes utilizadas (Figura 2.13) tienen una diferencia de hasta 2 veces: es decir, en este conjunto, la imagen más grande es el doble de grande que la imagen más pequeña.

Por último, de acuerdo con los resultados de las simulaciones, existe un 95 % de probabilidad de que el tiempo de procesamiento de una solicitud de redimensionamiento de una imagen de tamaño $1Mb \leq x \leq 2Mb$ tome 10,14 segundos o menos (Figura 2.22). En IM-Simple, al 100 % de las solicitudes le tomó 10 segundos o menos (Figura 2.19). Para este caso, el uso del tiempo promedio de procesamiento de solicitud de redimensionamiento podría ser utilizado para caracterizar el comporamiento de la función Lambda, pero, como en los dos casos anteriores, se considera que, el uso de la probabilidad acumulada sea más adecuado para brindar una predicciones más acertada.

Resultados Generales

En los tres experimentos propuestos, los resultados provenientes de las simulaciones lograron caracterizar en gran medida lo observado en IM-Simple. En 2.23 se muestran las 3000 simulaciones realizadas (1000 por cada caso). Las líneas punteadas verticales señalan en dónde se concentran el 95 % de los datos de las simulaciones: 1,6 segundos para las imágnes menores a 500Kb, 8 segundos para las imágenes de entre 500Kb a 1Mb y 10,14 segundos para las imágenes entre 1Mb y 2Mb.

En el Cuadro 2.4 se muestra un resumen estadístico de los tres experimentos llevados a cabo. En este resumen se evidencia los efectos del monitoreo de Amazon X-Ray en la versión IM-XRay de *Image Handler* en el tiempo promedio de procesamiento de una solicitud de redimensionamiento. Las diferencias

Probabilidad acumulada: solicitudes de redimensionamiento en imágenes $1Mb \leq x \leq 2Mb$ en PCM

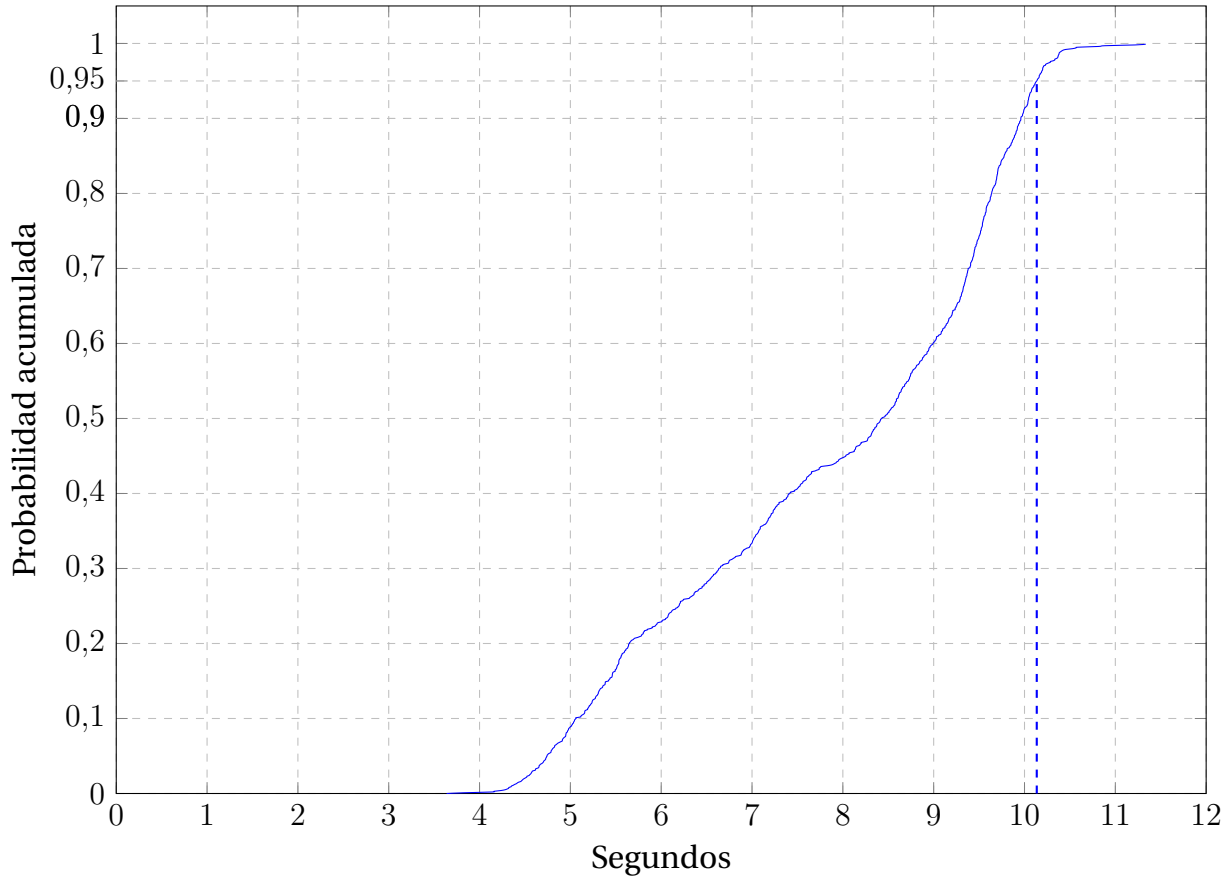

Figura 2.22: Probabilidad acumulada de solicitudes de redimensionamiento en imágenes $1Mb \leq x \leq 2Mb$ en PCM

3000 Simulaciones: Función de probabilidad acumulada para los tres escenarios de pruebas

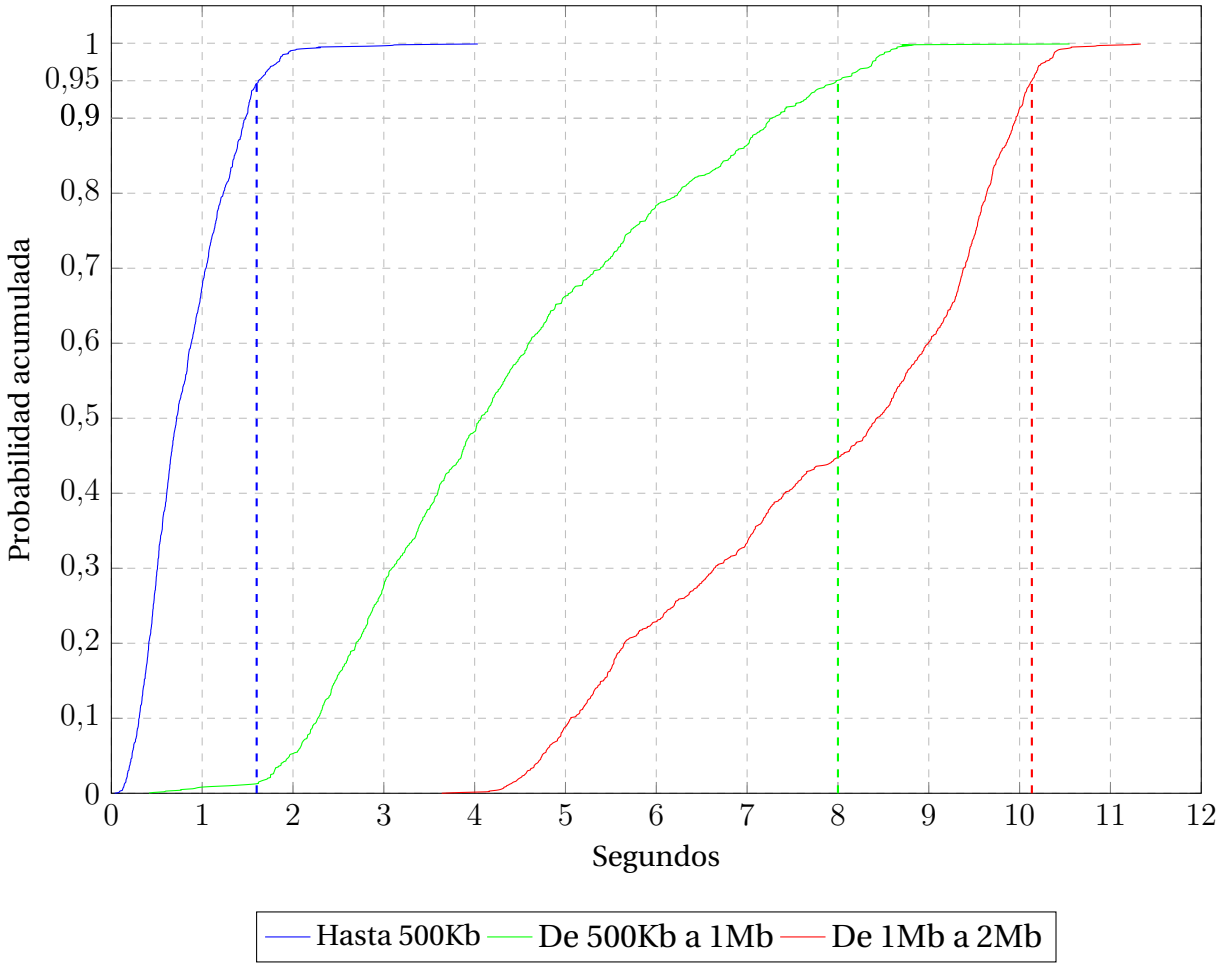

Figura 2.23: 3000 Simulaciones: Función de probabilidad acumulada para los tres escenarios de pruebas

entre el tiempo promedio de procesamiento en IM-Simple y IM-XRay fue de 209.965ms, 274.428ms y 257.773 para el experimento #1, #2 y #3 respectivamente. En promedio 247.389ms de diferencia. La desviación estándar de estos tres valores es de 33.463ms y el coeficiente de variación es de 0,13. Estos datos son muy similares entre sí, y esto hace que para el caso de las simulaciones en PCM se pueda argumentar que en promedio se puede esperar diferencia de 247.389ms en el tiempo de respuesta de las simulaciones *versus* los reportados por IM-Simple.

Cabe destacar que, a pesar que la posición acerca de las diferencias observadas entre los tiempos de respuesta de las simulaciones con IM-Simple, se hizo con base en tres valores, estos valores fueron obtenidos a partir de 3000 mediciones bajo tres grupos distintos de imágenes.

Desde el punto de vista de la arquitectura de software, *Image Handler* muestra mejores rendimientos cuando se prueba con imágenes iguales o menores a 500Kb que con aquellas mayores a ese tamaño. Dentro de las mediciones a los componentes de *Image Handler*, el que experimentó una mayor variablidad en los tiempos de procesamiento fue el componente de redimensionamiento. Aunque es el componente más importante dentro de esta arquitectura, es también el cuello de botella. Este es el principal componente a ser modificado en aras de probar diferentes comportamientos de *Image Handler*.

Varias otras optimizaciones podrían ser tomadas en cuenta aparte de valoras cambios en la biblioteca de redimensionamiento. Aunque puedan existir bibliotecas que logren disminuir en gran medida el redimensionamiento de una imagen, se hace útil también el considerar otras herramientas o estrategias para mejorar los tiempos de respuesta de la función por ejemplo:

1. Limitar el tamaño de las imágenes a procesar: una vez conocido el rendimiento de la función, se podría en limitar las imágenes a procesar basado en su tamaño, por ejemplo, solamente imágenes menores a 500Kb.

2. Utilizar una memoria intermedia de acceso rápido, un *caché*, para guardar copias de imágenes redimensionadas: luego de una primera solicitud de redimensionamiento, se guarda la imagen resultante en un *caché* y luego, entregar esta imagen en subsecuentes solicitudes de redimensionamiento. De esta forma las solicitudes repetitivas no llegarían a ser procesadas por la función Lambda. Esta estrategia se muestra en la Figura 2.1.

3. Preprocesar imágenes de gran tamaño fuera de línea: en el caso en donde se cuente con las imágenes originales de antemano, podría ser muy útil ejecutar procesos fuera de línea para redimensionar estas imágenes a los tamaños requeridos y hacer que *Image Handler* solamente sirva estas imágenes y no aplicar ningún tipo de lógica de redimensionamiento.

Hasta 500Kb			
Solicitud de redimensionamiento	IM-Simple	Palladio	Diferencia
Tiempo promedio	583.842ms	793.808ms	209.965ms
Desviación estándar	460.659ms	465.441ms	4.782ms
Varianza	212206.961	216635	–
Mediana	466.715ms	680.482ms	. –
Coeficiente de variación	0.987	0.683	–
Entre 500Kb a 1Mb			
Tiempo promedio	4073.600ms	4348.029ms	274.428ms
Desviación estándar	1731.974ms	1844.893ms	112.919ms
Varianza	2999736.844	3403633.84	–
Mediana	3658.825ms	3989.406ms	–
Coeficiente de variación	0.473	0.462	–
Entre 1Mb y 2Mb			
Tiempo promedio	7539.139ms	7796.913ms	257.773ms
Desviación estándar	1816.152ms	1914.258ms	98.106ms
Varianza	3298410.017	3664385	–
Mediana	8200.875ms	8310.293ms	–
Coeficiente de variación	0.221	0.230	–

Tabla 2.4: Resumen de datos estadísticos de los tres experimentos propuestos

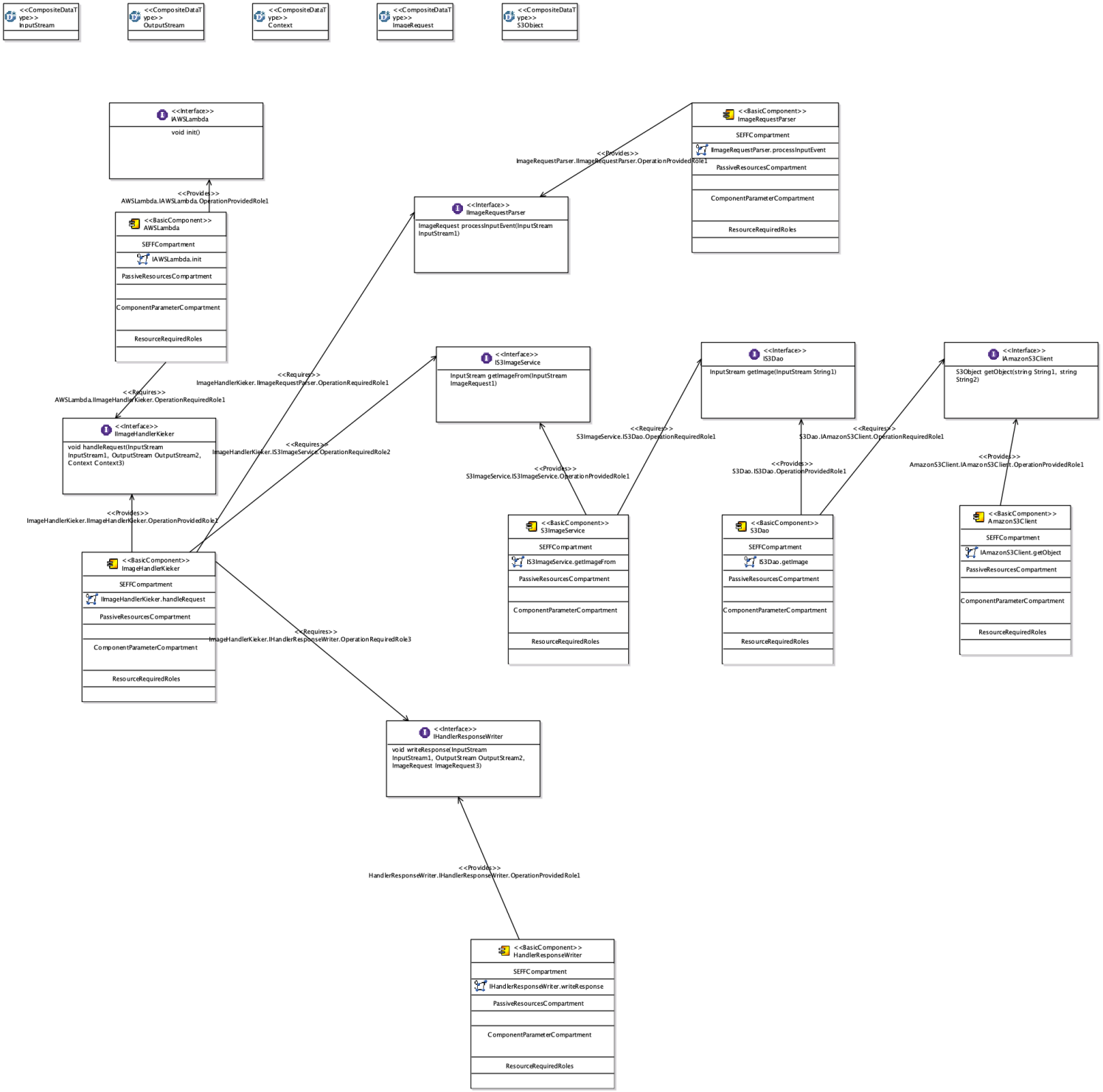

Figura 2.24: *Image-Handler* publicando eventos de rendimiento al servicio AWS X-Ray

2.4.2. Ejecución Secuencial Ininterrumpida de solicitudes de redimensionamiento

En el experimento de la Sección 2.4.1 se realizaron múltiples solicitudes de redimensionamiento con el fin de extraer y crear un modelo de rendimiento, ejecutar simulaciones sobre este modelo y comparar los resultados con lo observado en una versión de *Image Handler* real.

Una de las principales inquietudes durante la ejecución de funciones en la nube es de determinar si la velocidad en la que se procesa una solicitud resulta ser baja y muestren una tendencia predecible. Los desarrolladores e implementadores necesitan evaluar esto para afinar el código fuente yla arquitectura para evitar cobros elevados por el uso de la plataforma. De acuerdo con [8], una función puede pasar por dos grandes estados: uno “fría” y una “caliente”. En el estado frío, la plataforma que soporta la función en la nube debe aprovisionar los recursos necesarios para ejecutar la función. Durante esta fase se pueden experimentar los tiempos de respuesta más prolongados. La fase caliente se da luego de que la plataforma que soporta la función reconoce que la función está siendo invocada constantemente y que, debido a este uso, necesita proporcionarle mayores recursos computacionales para brindar mejores tiempos de respuesta. Se espera que, en una función que se esté invocando constantemente, solamente un porcentaje muy bajo del tiempo se encuentre en estado frío y la mayor parte (mientras continúe siendo invocada) estará en estado caliente.

Ejecución de solicitudes de redimensionamiento simultáneas para imágenes de tamaño menor a 500Kb

Para la realización de este experimento se contó con la siguiente configuración:

- *Sujeto de prueba:* La función Lambda IM-Simple
- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño menor o igual a 500Kb alojadas en Amazon S3.
- *Carga de trabajo:*
    1. 1000 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-Simple.
    2. 1000 invocaciones secuenciales de redimensionamient de una sola imagen a IM-Simple.
- *Herramientas de medición:* Amazon Cloudwatch.

En la carga de trabajo #1, utilizando imágenes aleatorias, la ejecución de las 1000 solicitudes de redimensionamiento fueron las mismas utilizadas para el experimento de la Sección 2.4.1.

La ejecución de las 1000 solicitudes de redimensionamiento tomó 2 horas y 36 minutos. Para la carga de trabajo #2, utilizando una sola imagen, las 1000 solicitudes de redimensionamiento tomaron 25 minutos. Se modificó el *script* en `Bash` del experimento de la Sección 2.4.1 para elegir aleatoriamente una imagen del *cluster* de 1000 y generar 1000 solicitudes secuenciales a partir de esta con dimensiones de ancho y alto estáticas.

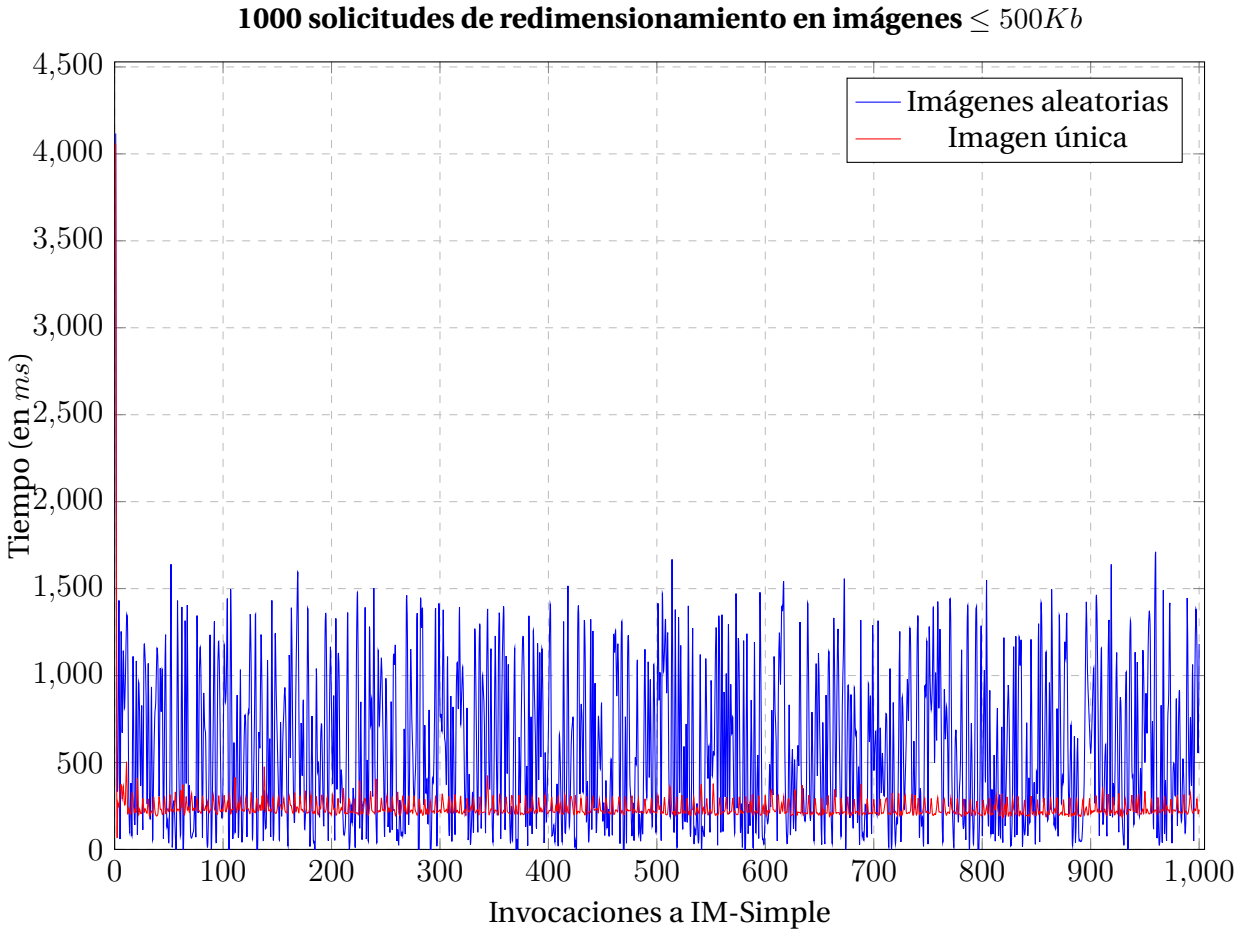

Figura 2.25: 1000 solicitudes de redimensionamiento en imágenes $\leq 500Kb$. La línea azul representa las solicitudes hechas utilizando imágenes aleatorias. La línea roja, las solicitudes utilizando una única imagen.

En la figura 2.25 se muestran 1000 ejecuciones secuenciales utilizando imágenes aleatorias (línea azul) y utilizando una sola imagen (línea roja). La imagen de la línea roja es de tamaño 40Kb. Los resultados de los tiempos de respuesta muestran una tendencia marcada: la primer solicitud tomó mucho más tiempo que el resto. Una vez realizada la primer solicitud, los tiempos de respuesta de la función mejoraron considerablemente y no se observaron en las subsecuentes invocaciones tiempos de respuesta que llegaran a ser similares al primero.

Ejecución de solicitudes de redimensionamiento simultáneas para imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb

Para la realización de este experimento se contó con la siguiente configuración:

- *Sujeto de prueba:* La función Lambda IM-Simple
- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño mayor a 500Kb y menor o igual a 1Mb alojadas en Amazon S3.
- *Carga de trabajo:*
    1. 1000 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-Simple.
    2. 1000 invocaciones secuenciales de redimensionamient de una sola imagen a IM-Simple.
- *Herramientas de medición:* Amazon Cloudwatch.

En la carga de trabajo #1, utilizando imágenes aleatorias, la ejecución de las 1000 solicitudes de redimensionamiento fueron las mismas utilizadas para el experimento de la Sección 2.4.1.

La ejecución de las 1000 solicitudes de redimensionamiento tomó 2 horas y 36 minutos. Para la carga de trabajo #2, utilizando una sola imagen, las 1000 solicitudes de redimensionamiento tomaron 2 horas y 30 minutos. Se modificó el *script* en `Bash` del experimento de la Sección 2.4.1 para elegir aleatoriamente una imagen del *cluster* de 1000 y generar 1000 solicitudes secuenciales a partir de esta con dimensiones de ancho y alto estáticas.

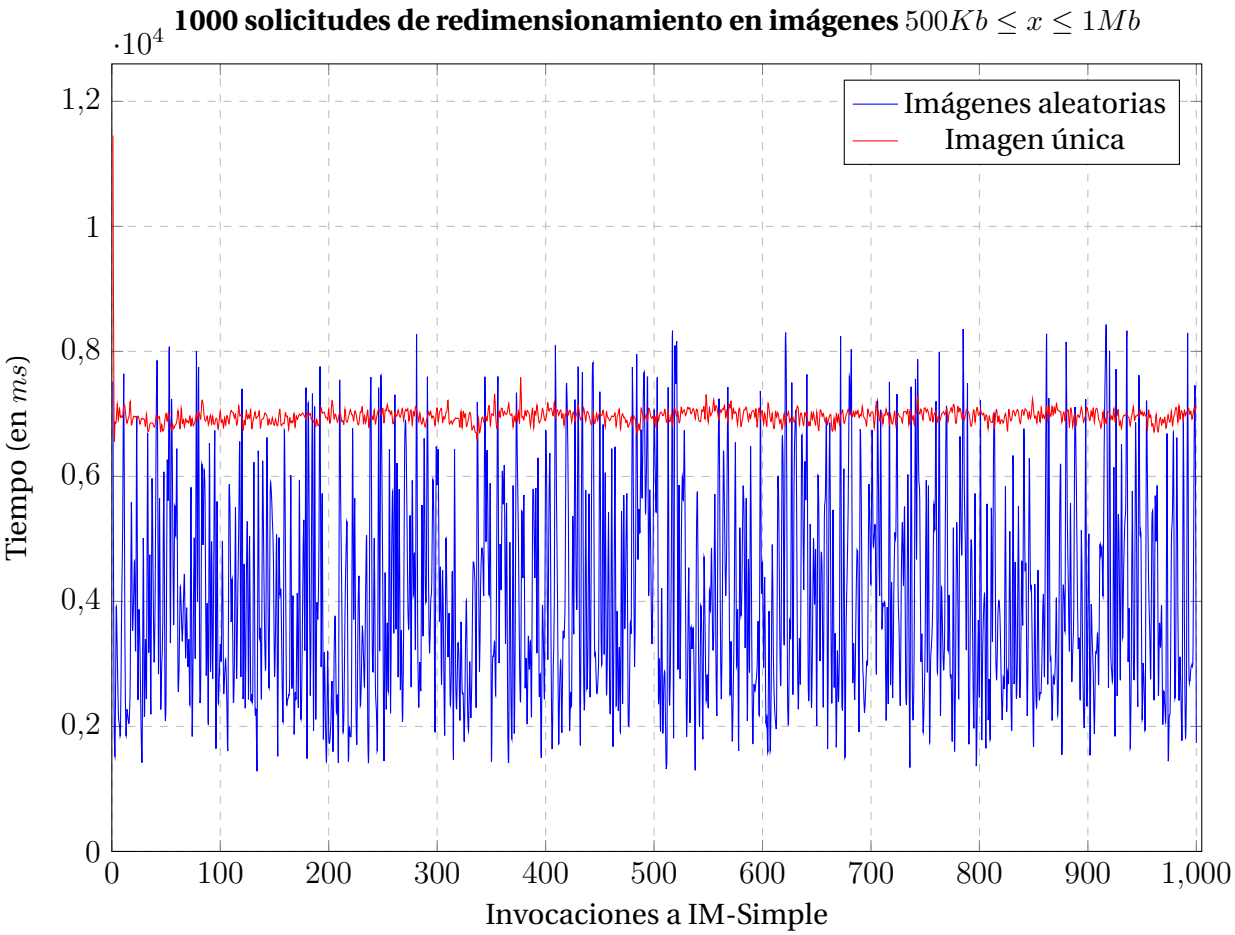

Figura 2.26: 1000 solicitudes de redimensionamiento en imágenes $500Kb \leq x \leq 1Mb$. La línea azul representa las solicitudes hechas utilizando imágenes aleatorias. La línea roja, las solicitudes utilizando una única imagen.

En la figura 2.26 se muestran 1000 ejecuciones secuenciales utilizando imágenes aleatorias (línea azul) y utilizando una sola imagen (línea roja). La imagen de la línea roja es de tamaño 750Kb. Los resultados de los tiempos de respuesta muestran una tendencia marcada: la primer solicitud tomó mucho más tiempo que el resto. Una vez realizada la primer solicitud, los tiempos de respuesta de la función mejoraron considerablemente y, en el caso de las solicitudes de redimensionamiento en imágenes aleatorias, sí se logró observar un caso en el que el tiempo de respuesta fue similar al observado en la primera solicitud. En el caso de la imagen única no se observaron tiempos de respuesta que llegaran

a ser similares al primero en posteriores invocaciones.

Ejecución de invocaciones de redimensionamiento simultáneas para imágenes de tamaño mayor a 1Mb y menor o igual a 2Mb

Para la realización de este experimento se contó con la siguiente configuración:

- *Sujeto de prueba:* La función Lambda IM-Simple
- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño mayor a 1Mb y menor o igual a 2Mb alojadas en Amazon S3.
- *Carga de trabajo:*
    1. 1000 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-Simple.
    2. 1000 invocaciones secuenciales de redimensionamiento de una sola imagen a IM-Simple.
- *Herramientas de medición:* Amazon Cloudwatch.

En la carga de trabajo #1, utilizando imágenes aleatorias, la ejecución de las 1000 solicitudes de redimensionamiento fueron las mismas utilizadas para el experimento de la Sección 2.4.1.

La ejecución de las 1000 solicitudes de redimensionamiento tomó 2 horas y 36 minutos. Para la carga de trabajo #2, utilizando una sola imagen, las 1000 solicitudes de redimensionamiento tomaron 2 horas y 16 minutos. Se modificó el *script* en `Bash` del experimento de la Sección 2.4.1 para elegir aleatoriamente

una imagen del *cluster* de 1000 y generar 1000 solicitudes secuenciales a partir de esta con dimensiones de ancho y alto estáticas.

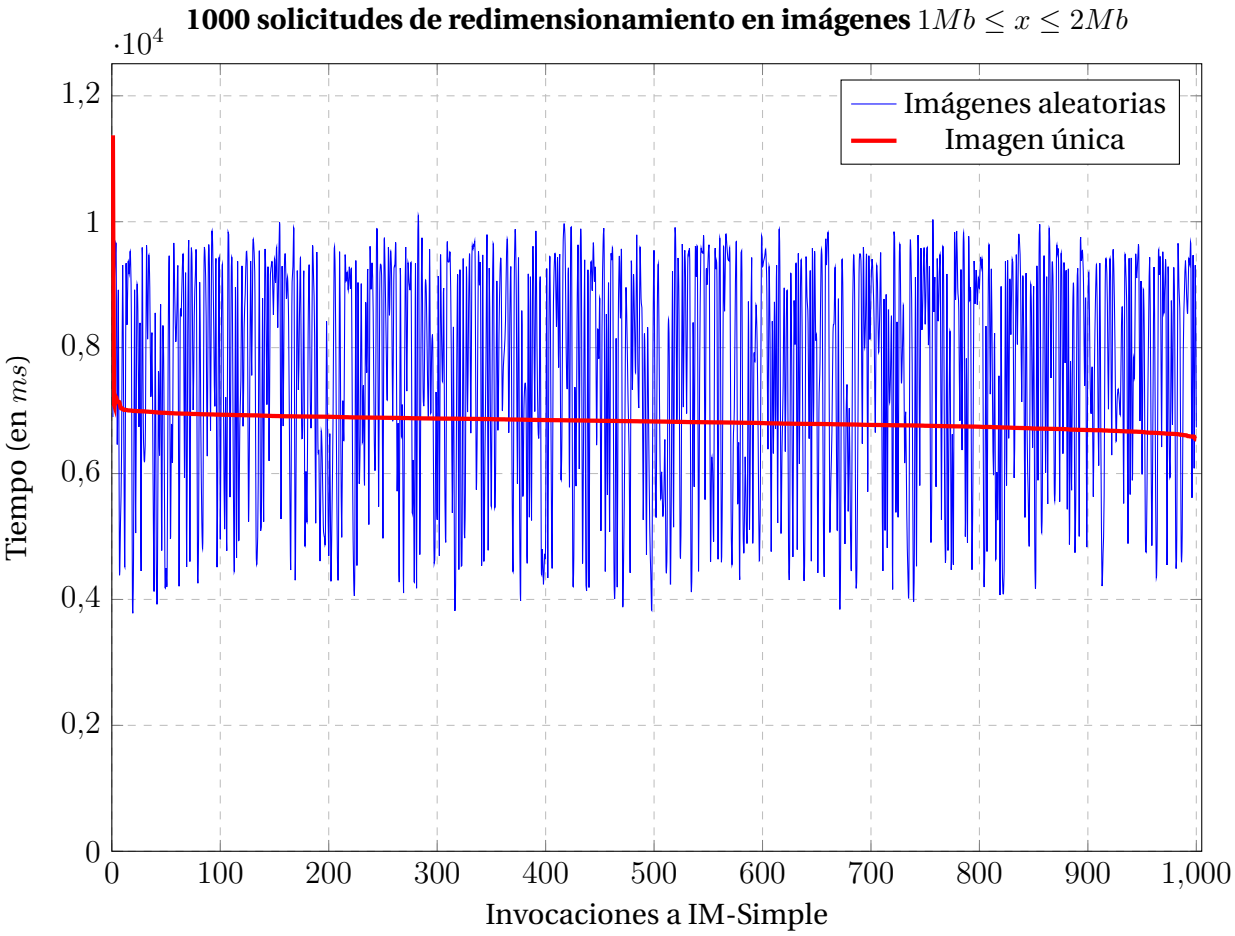

Figura 2.27: 1000 solicitudes de redimensionamiento en imágenes $1Mb \leq x \leq 2Mb$. La línea azul representa las solicitudes hechas utilizando imágenes aleatorias. La línea roja, las solicitudes utilizando una única imagen.

En la figura 2.27 se muestran 1000 ejecuciones secuenciales utilizando imágenes aleatorias (línea azul) y utilizando una sola imagen (línea roja). La imagen de la línea roja es de tamaño 1,4Mb. Los resultados de los tiempos de respuesta muestran una tendencia similar a los casos anteriores: la primer solicitud tomó mucho más tiempo que el resto. Una vez realizada la primer solicitud, los tiempos de respuesta de la función mejoraron considerablemente y, en el caso de las solicitudes de redimensionamiento en imágenes aleatorias, sí se logró observar

un caso en el que el tiempo de respuesta fue similar al observado en la primera solicitud. En el caso de la imagen única no se observaron tiempos de respuesta que llegaran a ser similares al primero en posteriores invocaciones.

Análisis de resultados

Luego de las invocaciones de redimensionamiento tanto en imágenes aleatorias como en una sola imagen, se pudo observar que, si bien en las primeras invocaciones se experimenta mayores tiempos de respuesta, los tiempos de respuesta entregados por la función Lambda lograron bajar significativamente y en los casos de las invocaciones de redimiensionamiento en una sola imagen, estos tiempos lograron mantenerse sumamente estables.

Este comportamiento sugiere que en la invocación de una función Lambda en AWS, la plataforma subyacente primero necesita aplicar tareas de aprovisionamiento para poner a disposición la función, y, una vez que esto se realiza, la función paulatinamente pasando a un estado "caliente" conforme van arribando las invocaciones de redimensionamiento. Aunque la arquitectura del servicio AWS Lambda no se encuentra disponible públicamente, en árticulos en Internet y en proyectos paralelos como SAM Cli y *AWS Lambda container image converter tool*[10] se sugiere el uso de contenedores Docker que son provisionados con el código de la función y que se encargan de procesar las invocaciones entrantes.

Relación con el modelo de rendimiento obtenido El modelo de rendimiento obtenido y las simulaciones hechas son agnósticos a los estados "frío" o "caliente" que se muestran en las observaciones. El principal objetivo de los simula-

[10] `https://github.com/awslabs/aws-lambda-container-image-converter`

dores de PCM es el de obtener múltiples combinaciones de ejecuciones de una arquitectura de software dada, y, a menos que una comportamiento o característica de esta logre ser introducido en el modelo es que su impacto debería de verse reflejado en las simulaciones.

Si bien, en el modelo obtenido no se introdujo esto explícitamente, las mediciones muestran que los tiempos de respuesta más prolongados se corresponden con el momento en que la función estaba en estado "frío", por ejemplo, durantes las primeras invocaciones. Si se toma como referencia el primer experimento ejecutado en la Sección 2.4.1, invocaciones de redimensionamiento en imágenes menores o iguales a 500Kb, en la figura 2.11, la primera medición a IM-Simple dió como resultado 4.1s y las subsecuentes 999 invocaciones bajaron a 1,6 segundos o menos; muchas de ellas incluso fueron menores a medio segundo. En los resultados de este experimento, existe un 95 % de probablidades que los tiempos de respuesta de las invocaciones sean de 1,6 segundos o menos, lo que, visto desde otro ángulo, quiere decir que existe un 5 % de probabilidad que los tiempos de respuesta de las invocaciones tomen entre 1,6 y 4 segundos en ser procesadas. Ahora bien, aunque 5 % parece ser una probabilidad muy alta y que no refleja lo visto en las mediciones, cuando se vuelve a los resultados de las simulaciones se observa que hay probabilidad mayor al 99 % de que una invocación de redimensionamiento tenga un tiempo de respuesta igual o menor a 2 segundos. Esto quiere decir que hay una probabilidad de menos del 1 % de que una invocación a la función Lambda tome entre 2 y 4 segundos en ser procesada. En el caso de las simulaciones de la Figura 2.11 solamente 6 casos muestran tiempos de respuesta de entre 2 y 4 segundos lo que representa un 0.006 % del total de las simulaciones realizadas.

Lo anterior señala que, para el caso de *Image Handler*, las simulaciones con

los tiempos de respuesta más prolongados representan una cantidad muy pequeña de la totalidad de los casos y que, estos casos se pueden convertir en potenciales escenarios de la función Lambda en estado frío. Las simulaciones muestran que a pesar del impacto que tienen estos escenarios en las invocaciones, estos realmente no llegan afectar el comportamiento general de la función Lambda.

Con respecto a los resultados de este experimento y su relación con los escenarios de invocaciones de redimiensionamiento en imágenes de tamaño de 500Kb y menor o igual a 1Mb, e imágenes de tamaño de 1Mb y menor a 2Mb, expuestas en la Sección 2.4.1, se puede emplear un análisis similar: las simulaciones que reportan los tiempos de respuesta más prolongados tienen el potencial de caracterizar las ejecuciones de la función en estado “frío”.

2.4.3. Variando el intervalo de las invocaciones de redimensionamiento

Este experimento se plantea como una variación del expuesto en la Sección 2.4.2, donde se evaluaba el efecto de las invocaciones simultáneas a la función Lambda. Acá se plantea variar el intervalo entre invocaciones a la función Lambda con el fin de valorar los efectos en el comportamiento de la función y si el modelo obtenido en la Sección 2.4.1 contribuye a explicar algo de lo observado; en particular si los intervalos entre las invocaciones hacen que el rendimiento de la función se considere “fría” o “caliente” en algún punto.

Estrategia de intervalo de invocaciones a *Image Handler*

Para evaluar los efectos de los intervalos de lanzamiento en las invocaciones a *Image Handler*, se propone la ejecución de un número de invocaciones simultáneas (una ráfaga), luego entrar en un tiempo de espera y luego, aplicar otra ráfaga de invocaciones.

El tiempo de espera entre ráfagas se irá incrementando al doble del tiempo de espera anterior. Por ejemplo, si el primer tiempo de espera es de 2 minutos, el siguiente será de 4 minutos, luego 8, 16, 32 minutos y así sucesivamente. Para este experimento se propone utilizar tiempos de espera inicial de 10 minutos para luego aumentarlo en subsecuentes ráfagas de invocaciones.

Ejecución de ráfagas de invocaciones de redimensionamiento para imágenes de tamaño menor a 500Kb

Para la realización de este experimento se utilizó la siguiente configuración base:

- *Sujeto de prueba:* La función Lambda IM-XRay.
- *Repositorio de imágenes: Cluster* de 1000 imágenes de tamaño menor o a 500Kb alojadas en Amazon S3.
- *Carga de trabajo:*
    - 50 invocaciones secuenciales de redimensionamiento de imágenes con dimensiones aleatorias a IM-XRay seguida de un tiempo de espera inicial de 10 minutos.

- Luego se ejecutará otra ráfaga de 100 invocaciones y se tendrá un tiempo de espera de 20 minutos.
- Se continuará aplicando la carga de trabajo anterior hasta que se logre observar algún cambio en la función.

- *Herramientas de medición:* Amazon Cloudwatch y Amazon X-Ray.

De acuerdo con lo observado en los experimentos anteriores, luego de la primera invocación, las subsiguientes hacen que la función entre en un estado "caliente". Por esta razón, ejercitar la función con una ráfaga de 100 invocaciones haría que se logre llegar a este estado fácilmente.

A diferencia de los experimentos anteriores, se utiliza la versión de IM-XRay en lugar de IM-Simple. La razón es que se quiere sacar provecho a las herramientas de monitoreo de AWS X-Ray para determinar si la función pasa de un estado *frío* $\rightarrow$ *caliente* $\rightarrow$ *frío* y determinar el impacto de este cambio en el tiempo de respuesta.

Variando el intervalo de las invocaciones de redimensionamiento en imágenes $\leq 500Kb$

La Figura 2.28 muestra 400 invocaciones de redimensionamiento en imágenes menores a 500Kb divididas en 100 ráfagas cada una. En la primer ráfaga la función se encuentra en estado "frío", sucede el aprovisionamiento inicial. Luego de esta primer invocación la función va entrando paulatinamente en estado "caliente" y se logra observar que para la mayoría de los casos las invocaciones de redimensionamiento no llegan a tomar más de 1,5 segundos. El tiempo promedio de ejecución de una ráfaga fue de 2 minutos y 45 segundos.

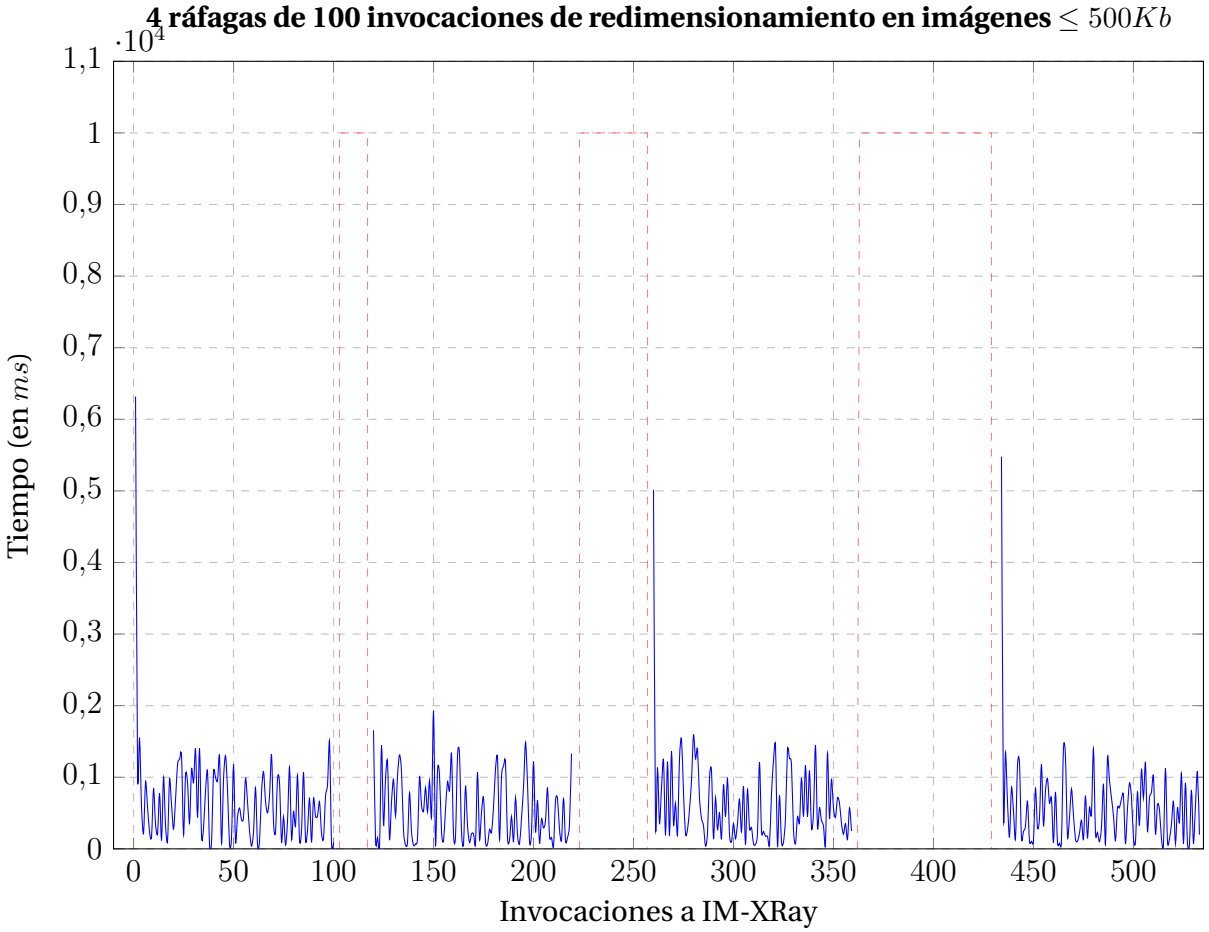

Figura 2.28: El espacio delimitado por la línea punteada roja representa 10 minutos de inactividad entre la primer ráfaga y la segunda. El segundo espacio delimitado por la línea punteada roja representa 20 minutos de inactividad entre la segunda y la tercera ráfaga.

En la segunda ráfaga, luego de 10 minutos, la función no experimenta un tiempo de respuesta inicial similar al de la primer ráfaga. Esto quiere decir que pasados 10 minutos de inactividad entre la primer y segunda ráfaga, la función continua en estado caliente.

En la tercer ráfaga, luego de 20 minutos de haberse ejecutado la última invocación de redimensionamiento de la ráfaga anterior, sí se llega observar un tiempo de respuesta inicial alto. AWS X-Ray reporta que en esta invocación inicial se tuvo que re-aprovisionar la función, lo que sugiere que, durante los 20 minutos de inactividad entre una ráfaga y otra, la función fue pasando progresivamente a un estado "frío" debido a que la plataforma pudo determinar que el nivel de actividad de la función (cantidad de invocaciones) fue decreciendo hasta un punto en donde no detectó actividad alguna.

La última ráfaga también experimentó un tiempo de respuesta inicial alto. Esta vez un poco mayor al de la ráfaga anterior pero menor a la de la primer ráfaga. Hubo 40 minutos de inactividad entre la tercera ráfaga y la cuarta.

Variando el intervalo de las invocaciones de redimensionamiento en imágenes $500Kb \leq x \leq 1Mb$

Se hizo el mismo ejercicio que en la Sección 2.4.3. La Figura 2.29 muestra los resultados de la ejecución de 5 ráfagas de 100 invocaciones cada una. El tiempo promedio de ejecución de una ráfaga fue de 9 minutos y 15 segundos. En la primera invocación de la primer ráfaga se obtuvo un tiempo de respuesta alto (función en estado frío). En la segunda ráfaga no se observó tiempos de respuesta elevados debido a que la función se encontraba en estado "caliente".

En la primera invocación de la tercera y cuarta ráfaga, AWS X-Ray reportó

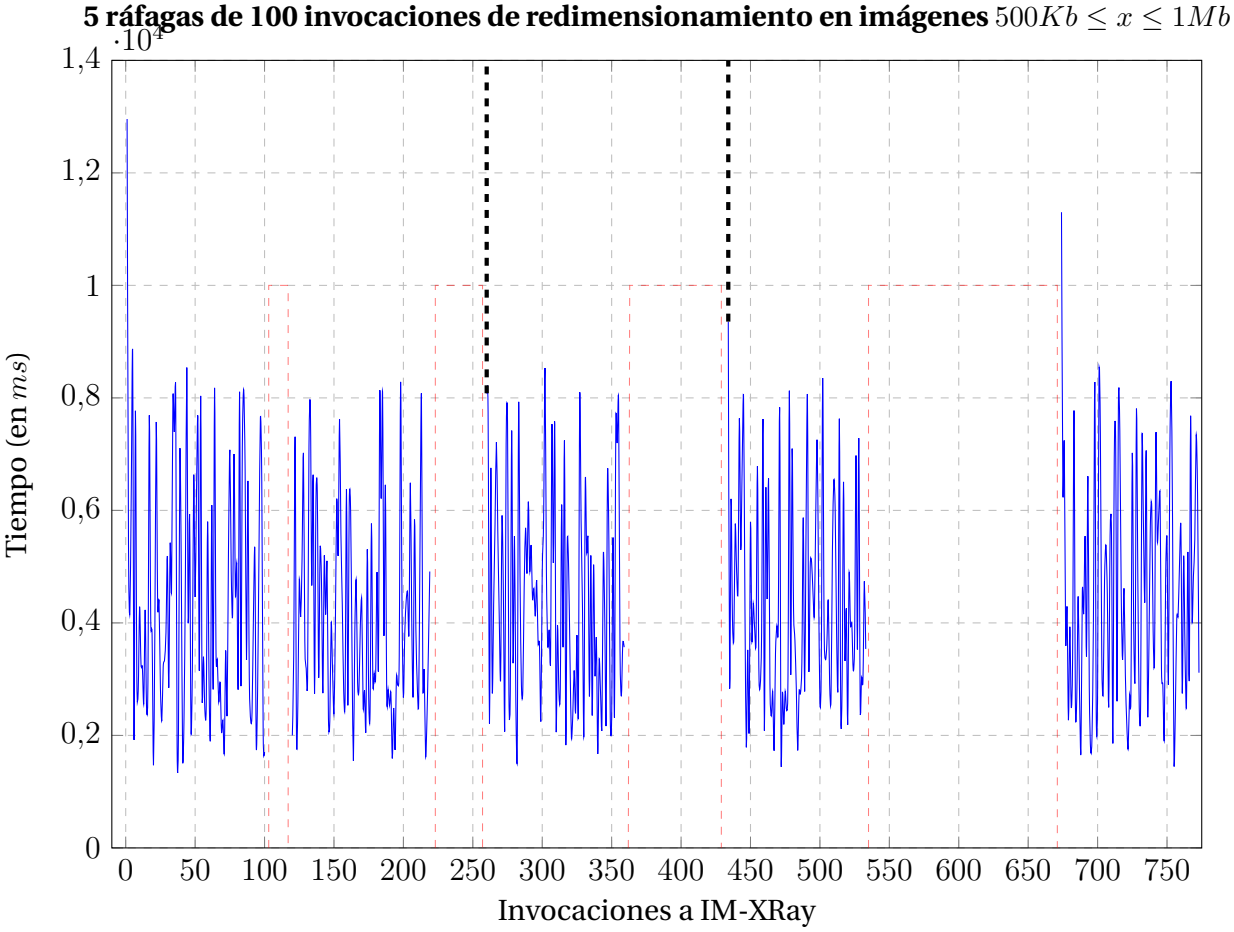

Figura 2.29: El espacio delimitado por la línea punteada roja representa 10 minutos de inactividad entre la primer ráfaga y la segunda. El espacio delimitado por la línea punteada verde representa 20 minutos de inactividad entre la segunda y la tercera ráfaga.

Entre 500Kb a 1Mb	
Ráfaga	Tiempo de 1º invocación
#1	13s
#2	N.A.
#3	8s
#4	9s
#5	11s

Tabla 2.5: Tiempo de respuesta de la primer invocación de redimensionamiento en cada ráfaga.

que en ambas hubo que re-aprovisionar la función. Es decir, el tiempo de inactividad entre la segunda ráfaga (20 minutos) y entre la tercera y la cuarta (40 minutos) fue suficiente para que la plataforma volviera a marcar la función como fría de nuevo. Lo interesante de estas dos primeras invocaciones, es que, a pesar que la plataforma tuvo que volver a re-aprovisionar la función, los tiempos de respuesta reportados fueron mucho más bajos que el de la invocación inicial de la primer ráfaga. En la Tabla 2.5 se pueden ver los tiempos de respuesta de las primeras invocaciones de cada ráfaga.

En la quinta ráfaga, el tiempo de respuesta de la primera invocación se elevó de nuevo. El tiempo de inactividad entre la cuarta ráfaga y la quinta fue de 80 minutos.

Debido a que los tiempos iniciales de la tercera y cuarta ráfaga fueron bajos (aunque AWS X-Ray reporta que para cada uno de ellos fue necesaria re-aprovisionar la función), fue que se la ejecución de una quita ráfaga se hizo necesaria con el fin de validar que, conforme los tiempos de inactividad entre ráfagas aumenta, el tiempo de respuesta de la primera invocación también aumenta.

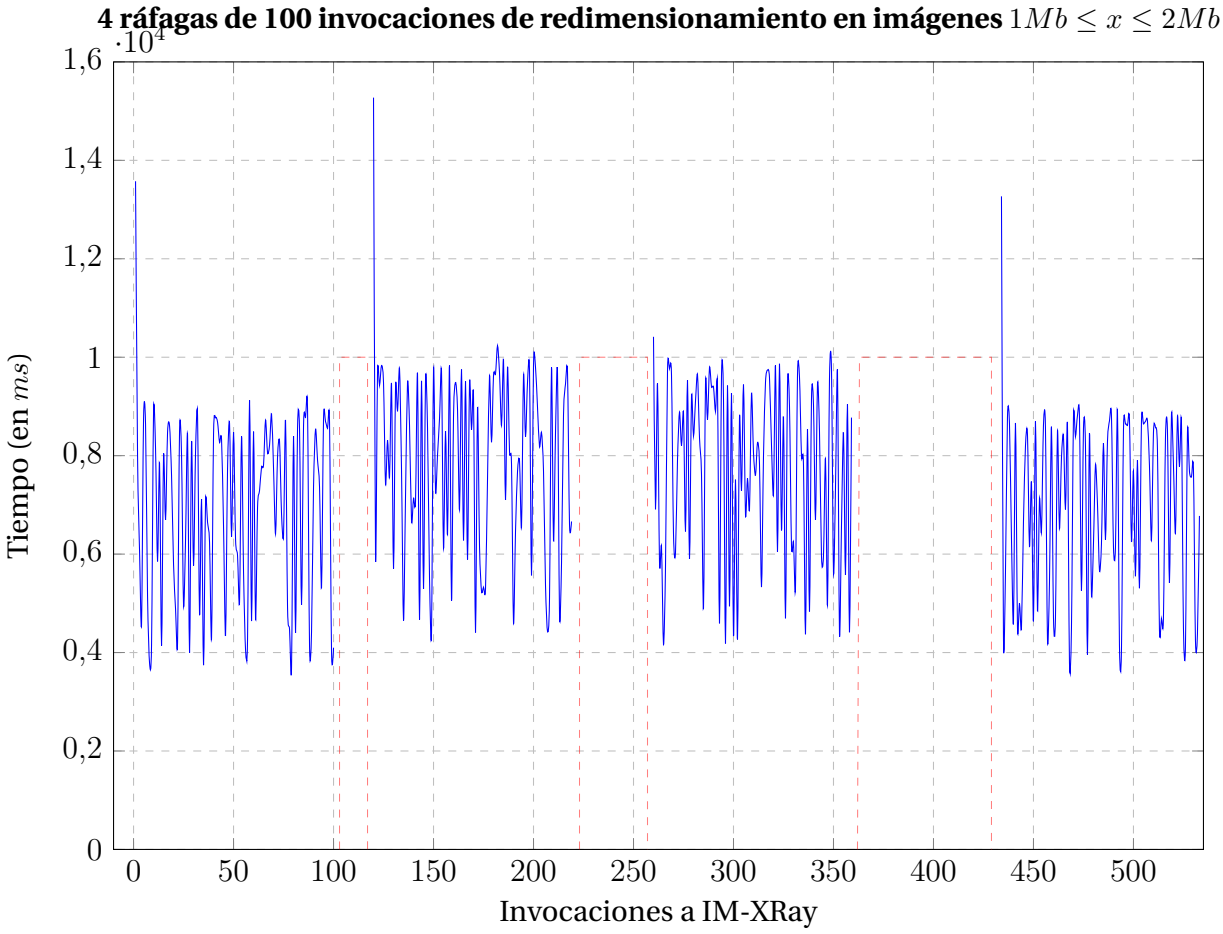

Figura 2.30: El espacio delimitado por la línea punteada roja representa 10 minutos de inactividad entre la primer ráfaga y la segunda. El espacio delimitado por la línea punteada verde representa 20 minutos de inactividad entre la segunda y la tercera ráfaga.

Variando el intervalo de las invocaciones de redimensionamiento en imágenes $1Mb \leq x \leq 2Mb$

Para este caso, el tiempo promedio de ejecución de una ráfaga fue de 13 minutos y 15 segundos. En la ejecución de cada ráfaga, AWS X-Ray reportó que fue necesario re-aprovisionar la función cuando se ejecutó la primera invocación de cada ráfaga. En la Figura 2.30 se muestran los resultados de la ejecución de las cuatro ráfagas.

Los tiempos de respuesta de las primeras invocaciones de cada ráfaga resul-

taron ser más similares entre sí y, también se pudo observar que cuando existen tiempos de inactivad más prolongados, como el que entre la ráfaga tres y la cuando que es de 40 minutos, a la siguiente invocación le toma un mayor tiempo de respuesta que la primer invocación de la ráfaga anterior.

Análisis de resultados

La principal observación que arrojó este experimento es que conforme van aumentando los tiempos de inactividad en la función Lambda, también van aumentando los tiempos de respuesta entregados por la primera invocación luego de este tiempo de inactividad.

De acuerdo con [9], una vez que una función ha sido instalada en la plataforma, la primer invocación debe de pasar por el proceso de aprovisionamiento. Luego de que esta primer invocación es procesada, la función pasa a un estado activo o "caliente" y el contenedor que la soporta se puede reutilizar para subsecuentes invocaciones. Cuando se detecta que la función se vuelve inactiva o "fría", el contenedor que la soporta se vuelve candidato a ser "reciclado" para ser utilizado por otra función que lo necesite.

Si bien, no hay un tiempo límite definido para decidir cuándo el contenedor va a ser reciclado o no, los resultados de este experimento y los expuestos en [9] y [10] demuestran que entre mayor sean los tiempos de inactividad de una función, mayor será la probabilidad de que el contenedor que la soporta sea reciclado y que, cuando se vuelva a invocar a la función, se experimente un tiempo de respuesta mayor al promedio debido al proceso de aprovisionamiento. Según [9], una vez que la función deja de recibir invocaciones y entra en un estado inactivo, el contenedor que la soporta se mantiene disponible por

al menos 25 minutos.

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. Boyd, "Serverless architecture: Five design patterns," March 2017. https://thenewstack.io/serverless-architecture-five-design-patterns/.

[4] A. W. Services, "AWS lambda - resources: Reference architectures," 2018. https://aws.amazon.com/lambda/resources/reference-architectures/.

[5] A. W. Services, "Serverless image handler – AWS answers," 2018. https://aws.amazon.com/answers/web-applications/serverless-image-handler/.

[6] J. Walter, S. Eismann, J. Grohmann, D. Okanovic, and S. Kounev, "Tools for declarative performance engineering," in *Companion of the 2018 ACM/S-*

PEC International Conference on Performance Engineering, ICPE '18, (New York, NY, USA), pp. 53–56, ACM, 2018.

[7] K. Project, "Kieker 1.13 user guide," Oct 2017. http://kieker-monitoring.net/documentation/, obtenido el 15 de Abril del 2019.

[8] 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.

[9] M. Shilkov, "Cold starts in aws lambda," Jun 2019. https://mikhail.io/serverless/coldstarts/aws/, obtenido el 10 de Julio del 2019.

[10] M. Shilkov, "When does cold start happen on aws lambda?," Jun 2019. https://mikhail.io/serverless/coldstarts/aws/intervals/, obtenido el 10 de Julio del 2019.