Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Mejora de un Sistema de Auto-escalado para Sistemas Distribuidos

Autor: Carlos Miguel Alonso
Tutor: Victor Rampérez Martín

Madrid, Febrero - 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Mejora de un Sistema de Auto-escalado para Sistemas Distribuidos

Febrero - 2022

Autor: Carlos Miguel Alonso
Tutor: Victor Rampérez Martín
Departamento de Lenguajes, Sistemas Informáticos e Ingeniería del Software
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Índice general

Índice de Listings

Índice de Tablas

Índice de Figuras

Resumen

Uno de los principales pilares de la computación en la nube es la elasticidad, es decir, la capacidad de ajustar los recursos necesarios de forma automática para cubrir la demanda en cada instante. Los sistemas de auto-escalado son los encargados de dotar de elasticidad a los sistemas escalables. Dichos sistemas de auto-escalado están recibiendo gran atención en los últimos años debido a la expansión de las arquitecturas basadas en micro-servicios (e.g. Kubernetes y Docker) y al auge de la computación en la nube.

En el contexto de este proyecto, se está desarrollando un sistema de auto-escalado que combina las técnicas de auto-escalado reactivas y predictivas mediante la aplicación de modelos estadísticos.

Este Trabajo de Fin de Grado aborda la mejora del auto-escalado de sistemas distribuidos por medio del estudio del comportamiento de E-SilboPs bajo una carga de trabajo real a través de la obtención y monitorización de las métricas de rendimiento relevantes.

Con este fin, se ha llevado a cabo una búsqueda de una carga de trabajo que se haya usado en un contexto de uso real. La carga escogida, procedente del sistema PADRES, un sistema publicador/subscriptor basado en contenido ampliamente usado en literatura, ha de ser traducida al formato de E-SilboPS. Para esto, se ha diseñado e implementado un generador de cargas de trabajo reales, que traducirá los eventos de dicha carga, manteniendo el contexto y contenido de cada uno de ellos (usuario, parámetros de las subscripciones y publicaciones, etc.).

Una vez traducida la carga de trabajo, para medir el rendimiento del sistema con esta, se han diseñado e implementado diferentes pruebas para obtener estas métricas de rendimiento, lo que permite identificar la saturación y comportamiento del sistema. Esto ha permitido la generación de otras cargas basadas en la real para llevar al sistema a situaciones especiales y medir su respuesta de forma más detallada.

A continuación, se han diseñado e implementado diferentes modelos predictivos basados en Machine-Learning y Deep-Learning, que se aplicarán a los resultados de las pruebas realizadas, con el objetivo de predecir el comportamiento del sistema y así aplicar estas predicciones al sistema de auto-escalado.

Los resultados obtenidos en este Trabajo de Fin de Grado muestran el correcto funcionamiento del generador de cargas de trabajo, al haber traducido la carga de PADRES de forma satisfactoria; y la utilidad de dicha carga, al comprobar que E-SilboPS llega a saturar parcialmente, posibilitando la aplicación de los modelos predictivos para predecir futuras situaciones de saturación.

Palabras Clave: Auto-escalado, Sistemas Distribuidos, Cloud, Modelos Estadísticos.

Abstract

One of the main pillars of cloud computing is elasticity, i.e. the ability to automatically adjust the resources needed to meet the demand at any given time. Auto-scaling systems are responsible of providing elasticity to scalable systems. This auto-scaling systems are getting increasing attention in recent years thanks to the growth of micro-services based architectures, e.g. Kubernetes and Docker, and the rise on popularity of cloud computing.

In the context of this project, an auto-scaler system is being developed, that combines both reactive and predictive auto-scaling techniques by applying statistical models.

This Trabajo de Fin de Grado addresses the improvement of the auto-scaling of distributed systems by studying the behaviour of E-SilboPS under a real-world workload, obtaining the relevant performance measurements.

To this end, a search has been carried out for a workload that has been used in a real-world context. This workload must be compatible with the E-SilboPS system, analysing its content to obtain statistical data, and verifying the completeness and compatibility of this workload data.

The chosen real-world workload, extracted from PADRES, a content-based publish/-subscribe system widely used on literature; must be translated to the E-SilboPS format. To achieve this, a real workload generator has been designed and implemented, that performs this translation by interpreting each event and its content, and transforming it to the E-SilboPS syntax, keeping its original context (user, subscription and publications parameters, etc.).

With the workload already translated, to test the system performance using it, different tests have been developed to obtain its system performance metrics, allowing detection of situations in which the system saturates. This allowed the creation of other workloads, based on the real-world one, that tests the system on special situations and measure the system response in more detail.

Once the system has been tested, different predictive models have been designed and implemented, based on Machine-Learning and Deep-Learning, that will be applied to the results of the tests carried out, in order to predict the behaviour of the system and then apply these predictions to the auto-scaling system.

The results obtained of this Trabajo de Fin de Grado reflect the real workload generator working correctly, since it translated the PADRES real-world workload; and the usefulness of said workload through verifying that E-SilboPS partially saturates, which opens the door to the possibility of implementing improvements on FLAS if new predictive models are applied.

Keywords: Auto-scaling, Distributed Systems, Cloud, Statistical Models.

Capítulo 1

Introducción

En este capítulo se presentan y explican la motivación, el contexto, los objetivos específicos del trabajo realizado y la estructura de este documento, con la intención de proporcionar una visión general del mismo.

1.1. Motivación del proyecto

En los últimos años se ha producido un incremento significativo en la digitalización de numerosos aspectos de la vida cotidiana de las personas como consecuencia de la popularización de nuevos paradigmas tecnológicos como las Smart Cities o el Internet of Things[1] (o IoT, por sus siglas).

Estos paradigmas se basan en la infraestructura proporcionada por los sistemas dirigidos por eventos, los cuales son de vital importancia para la transmisión de datos e información en tiempo real, como ocurre con el IoT, ya que proporcionan una infraestructura y arquitectura mediante Cloud Computing[2] capaz de proveer de los recursos necesarios a sistemas distribuidos que mueven grandes cantidades de datos (por ejemplo, sistemas publicador/subscriptor o sistemas distribuidos de procesamiento de datos).

La ventaja del Cloud Computing para los sistemas publicador/subscriptor es la elasticidad que proporciona a dicho sistema, es decir, la capacidad de adapta los recursos dedicados a este en base a sus necesidades de cómputo. Esta característica permite optimizar la utilización de los recursos de computación, ya los entornos de Cloud Computing usualmente, se basan en el pago bajo demanda (*pay-as-you-go*) de recursos dedicados.

Para aprovechar esta elasticidad, los sistemas desplegados en una infraestructura cloud, en concreto, los sistemas publicador/subscriptor, deben de implementar dicha elasticidad, para poder adaptar los recursos disponibles a la cantidad de trabajo entrante, al encontrarse en un entorno muy cambiante, donde puede haber subidas y caídas repentinas de trabajo, llevando al sistema a una desabastecimiento o sobreabastecimiento de recursos, respectivamente.

[1] Dispositivos físicos con sensores y software que conectan e intercambian datos con otros dispositivos y sistemas a través de Internet

[2] Servicios de computación a través de Internet

Esta elasticidad (o auto-escalado) se consigue mediante la implementación de un auto-escalador (auto-scaler) en el sistema publicador/subscriptor, que llevará a cabo las operaciones de escalado de forma automática y óptima, mediante la monitorización del uso de los recursos de computación, y para así saber cuánto ha de escalar (en número de recursos añadidos/quitados). Existen diferentes paradigmas de auto-escalado, que llevan a cabo las operaciones de escalado de diferentes formas.

El sistema usado y sobre el que se ha estado trabajando en este proyecto es E-SilboPS[1][2][3], un sistema publicador/subscriptor basado en contenido con cierto grado de elasticidad[3]. Este sistema es la continuación natural, mediante la implementación de la elasticidad y de numerosas mejoras, de SilboPS[3], un sistema publicador/subscriptor basado en contenido, e inspirado por SIENA[4].

El principal problema con el desarrollo de este sistema es la dificultad de probar su comportamiento en una situación real mediante el uso de una carga de trabajo con datos reales, ya que la publicación de unos datos de este estilo conllevan problemas de seguridad, tanto para la propia empresa como para los usuarios, ya que se expone la información sobre sus intereses, gustos, etc.

Con una carga de trabajo real, se puede analizar el comportamiento del sistema en un entorno real, con el objetivo de encontrar los puntos a mejorar, para aumentar la precisión de las predicciones y el escalado, siendo este el objetivo de este trabajo y proyecto.

El trabajo desarrollado en este TFG es la continuación natural de previos TFMs y tésis doctorales[1][2][3], los cuales desarrollaron y mejoraron el sistema E-SilboPS, usado en este TFG.

1.2. Contexto del proyecto

1.2.1. Sistemas publicador/subscriptor auto-escalables

Los sistemas publicador/subscriptor operan como mediadores entre las entidades que actúan de publicadores, que generan información que puede ser de interés para un usuario; y los propios usuarios, que especifican sus intereses mediante las subscripciones enviadas a este sistema, por lo que solo reciben la información que encaja en dichos intereses)[5][6]. Este proceso se lleva a cabo comparando cada una de las subscripciones que el sistema tiene almacenadas, y generando una lista de los usuarios a los que hay que enviar dicha publicación.

Existen dos tipos de paradigmas dentro de los sistemas publicador/subscriptor, los basados en tema (**topic-based**), que permiten al usuario recibir todas las publicaciones relacionadas con el tema especificado; y basados en contenido (**content-based**), que proporcionan al usuario más medidas para establecer qué quiere recibir por medio de especificar pares clave-valor que se comprobarán contra el contenido de la publicación.

Estos sistemas pueden ser auto-escalables, es decir, adaptan sus recursos y dedicación de los mismos a la cantidad de trabajo que están recibiendo. Este auto-escalado puede ser de dos tipos, **reactivo**, por el cual el sistema escala[4] cuando se detecta un

[3]Sistema que adapta sus recursos a la cantidad de trabajo entrante de forma automática.
[4]Aumento/Disminución de los recursos

elevado/bajo consumo de cierto recurso computacional, o **predictivo**, anticipando el aumento/disminución de recursos a la saturación/desuso de los mismos, maximizando la eficiencia del sistema y minimizando el malgasto.

1.2.2. E-SilboPS

Para desarrollar este proyecto, se ha utilizado, con el fin de poder poner en práctica las mejoras aplicables a los sistemas publicador/subscriptor, el sistema E-SilboPS, que es la continuación del trabajo de previas tesis doctorales [2][3], llevadas a cabo en el grupo de investigación *CETTICO* [7], en la Escuela Técnica Superior de Ingenieros Informáticos[5] de la Universidad Politécnica de Madrid[6].

El grupo de investigación *CETTICO* desarrolla y lleva proyectos en diferentes líneas de investigación, proporcionando soluciones tecnológicas a diferentes problemas en sistemas distribuidos, minería de datos y tecnologías web, entre otros muchos; tanto para empresas y sus departamentos de I+D+i como para proyectos de investigación, como este.

Este Trabajo de Fin de Grado se centra en la mejora de un sistema distribuido publish/subscribe auto-escalable basado en contenido, mediante el análisis de sus principales métricas de rendimiento y la aplicación de modelos predictivos, con el objetivo último de mejorar la predicción de los recursos que necesita, tanto en cantidad como en tipo, de forma que la eficiencia del sistema sea la máxima en cada momento.

1.3. Objetivos

El principal objetivo de este trabajo es la mejora de un sistema de auto-escalado desarrollado para sistemas distribuidos, en concreto, para sistemas de publicador/-subscriptor, mediante la implementación de modelos predictivos aplicados a cargas de trabajo reales.

De este objetivo general, se pueden extraer los siguientes más específicos, que reflejan de forma más concisa el desarrollo del proyecto:

- Realizar un análisis de la situación actual de los sistemas publicador/subscriptor auto-escalables, con el objetivo de conocer su funcionamiento y sus limitaciones, extrayendo los datos más relevantes, desarrollando una visión detallada y adquiriendo un conocimiento general de estos sistemas.
- Realizar un análisis de las cargas de trabajo actuales y en uso en los sistemas auto-escalables ya desarrollados, extrayendo la información más relevante sobre ellos para determinar su posible uso en futuras pruebas.
- Realizar un estudio de las cargas de trabajo actualmente en uso en sistemas distribuidos, y buscar una carga de trabajo real, que se haya obtenido por medio de un sistema real, y que sea compatible y exportable al sistema que se utilizará.
- Realizar un análisis estático de la carga de trabajo escogida, extrayendo los datos más relevantes, y comprobando su estructura y completa compatibilidad

[5] https://www.fi.upm.es/
[6] https://www.upm.es/

con el sistema que se utilizará, con el objetivo de traducir dicha carga a una carga compatible con el input del sistema a utilizar, por medio de desarrollar un traductor que produzca una carga de trabajo basada en la escogida, y sea compatible con la entrada del sistema a utilizar.

- Implementar pruebas que utilicen la carga de trabajo real escogida, con el objetivo de obtener las medidas de rendimiento relevantes del sistema, para analizar su comportamiento y obtener conclusiones de su eficiencia.
- Crear nuevas cargas de trabajo reales, partiendo de los resultados de las pruebas y de la carga de trabajo ya presente, de forma que dichas cargas lleven al sistema a diferentes situaciones cuyo estudio, y el sus medidas de rendimiento más relevantes, es de interés para el trabajo actual.
- Desarrollar e implementar modelos predictivos que se adapten a las diferentes situaciones a las que el sistema se puede enfrentar en un entorno real, basando dichos modelos en los resultados y pruebas llevadas a cabo previamente.

1.4. Estructura del Documento

Esta memoria, que cubre el trabajo realizado en este TFG, se divide en los diferentes apartados, los cuales cubren los objetivos establecidos en la *Sección 1.3 Objetivos*.

Tras esta introducción, en el *Capítulo 2 Trabajo relacionado y Estado del Arte*, se presenta el trabajo externo que se ha llevado a cabo previamente en este proyecto, así como la situación actual de los sistemas involucrados en el desarrollo e implementación del mismo, los problemas actuales de estos sistemas, y diferenciando entre los distintos tipos de sistemas de auto-escalado.

Una vez presentada la situación en la que se ha desarrollado este proyecto, en el *Capítulo 3 Desarrollo de un generador de cargas de trabajo para sistemas publicador/subscriptor basados en contenido*, se exponen las decisiones de diseño e implementación de dicho generador, los sistemas y herramientas que se han usado, y análisis de los resultados obtenidos de aplicar este generador a E-SilboPS, el sistema usado.

Con el objetivo de mejorar las predicciones del sistema, se presenta, en el *Capítulo 4 Generación de modelos predictivos para el auto-escalado de E-SilboPS*, el diseño y la implementación de los modelos predictivos para E-SilboPS, exponiendo el objetivo de estos modelos, su base y los resultados esperados de la aplicación de estos modelos.

A continuación, en el *Capítulo 5 Impacto del trabajo*, se relacionan las consecuencias y el impacto general de este proyecto, así como su relación con los Objetivos de Desarrollo Sostenible, y en cuáles se enmarca el mismo.

Por último, se presentan, en el *Capítulo 6 Resultados y conclusiones*, la información obtenida gracias a este proyecto y los resultados que ha generado. De igual manera, se explican las conclusiones personales del autor, así como el trabajo futuro del proyecto, enfocado a futuras mejoras del auto-escalado, la aplicación de los modelos predictivos, y mejoras en el sistema.

Capítulo 2

Trabajo relacionado y Estado del Arte

En esta sección se trata y expone el estado actual de las tecnologías utilizadas en este proyecto, así como la presentación de proyectos previos y similares a este, proporcionando una visión necesaria para la comprensión del trabajo realizado.

2.1. Sistemas publicador/subscriptor basados en contenido

2.1.1. Sistemas publicador/subscriptor

Los sistemas publicador/subscriptor siguen el paradigma de publicador/subscriptor, siendo los intermediarios entre los subscriptores (usuarios con interés en recibir cierta información) y los publicadores (los productores de dicha información)[8].

Su papel consiste en almacenar las subscripciones de los usuarios, y por cada publicación recibida, generar una lista de subscriptores interesados en esta y enviar dicha publicación a cada subscriptor de la lista. Este proceso de verificar que los intereses de un usuario (por medio de su subscripción) concuerdan con los parámetros de la publicación se denomina ”machear una publicación con una subscripción” (o, en su defecto, con las subscripciones del sistema).

Este paradigma ha permitido la distribución de las publicaciones de forma más eficiente ya que los usuarios solo reciben la información que desean, y los publicadores no tienen que manejar las subscripciones de dichos usuarios.

Esto se conoce como desacoplamiento entre subscriptores y publicadores, que se puede descomponer en las siguientes dimensiones[]:

- Desacoplamiento espacial: los subscriptores no tienen que conocerse entre si.
- Desacoplamiento temporal: los subscriptores no tienen que participar en el sistema de forma simultánea.
- Desacoplamiento de la sincronización: los subscriptores no tienen que estar activos en el sistema para recibir las notificaciones[1].

[1] Publicación que se ha enviado a un usuario al machear con su subscripción[5]

Los sistemas publicador/subscriptor presentan diferentes variantes, principalmente, en base al patrón usado para publicaciones y subscripciones[1]:

- Basados en tema: Las publicaciones se clasifican mediante una palabra clave que identifica el tema de la publicación. Los subscriptores especifican esta palabra clave en las subscripciones, de forma que reciban solo las notificaciones cuyo tema sea el especificado.
- Basados en contenido: Las subscripciones especifican los parámetros (contenido) de las publicaciones que el usuario quiere recibir.

A causa de esto, su utilización se ha popularizado en numerosas aplicaciones[5].

2.1.2. SIENA

SIENA [4], cuyo nombre proviene de las iniciales, en inglés, de Scalable Internet Event Notification Architectures, es un sistema publicador/subscriptor basado en contenido diseñado para maximizar la escalabilidad del sistema.

Una de las características de SIENA es la introducción de un nuevo tipo de mensaje, los avisos (o advertisements en inglés). Estos avisos son emitidos por los publicadores previo envío de las publicaciones, y sirven para informar al sistema del tipo de eventos que van a producir. SIENA también implementa técnicas para analizar las subscripciones que recibe el sistema, de forma que se minimice la propagación innecesaria de estas por el sistema

A causa de esto, SIENA se ha convertido en el marco de referencia para el diseño e implementación de sistemas publicador/subscriptor basados en contenido.

2.1.3. PADRES

PADRES[8] es un sistema publicador/subscriptor basado en contenido cuya arquitectura se basa en la comunicación punto a punto (*peer-to-peer (P2P)*) entre los usuarios del sistema.

Este sistema implementa algunas características adicionales, como un sistema de reglas de macheo y enrutamiento que se aplica a las publicaciones y subscripciones que llegan al sistema, de forma que se disminuye el tiempo de procesamiento y creación de rutas entre los publicadores/subscriptores y el sistema; subscripciones compuestas, las cuales se basan en la representación de una subscripción por medio de múltiples subscripciones atómicas[8][9][10].

PADRES ha sido usado en gran variedad de proyectos y casos de uso, debido a su versatilidad y eficiencia[11].

2.1.4. E-StreamHub

E-StreamHub[6] es sistema publicador/subscriptor con elasticidad[12], que extiende y mejora un sistema publicador/subscriptor escalable llamado StreamHub[13].

Al estar desarrollado a partir de StreamHub, E-Streamhub utiliza los operadores definidos en este, que están constituidos por diferentes componentes (como se puede ver en Figura 2.1), desplegando uno por cada host (o máquina) activo en el sistema, que son:

Access Point (AP)
Estos operadores manejan los eventos que recibe el sistema, dividiendo las subscripciones de forma equitativa entre los *Matchers* activos, mientras que las publicaciones se envían mediante broadcast a todos los *Matchers* activos (una copia a cada *Matcher*), lo que permite comparar las publicaciones con las subscripciones presentes en el sistema de forma paralela.

Matchers (M)
Los *Matchers* se encargan de comparar las publicaciones recibidas con su lista de subscripciones, generando una lista de subscripciones cuyos parámetros (intereses) coinciden con los de la publicación, que se enviará, junto con la propia publicación, al *Exit Point* correspondiente.

Exit Point (EP)
Cada *Exit Point* combina las listas de subscripciones generadas por todos los *Matchers*, y envía la publicación a cada uno de los subscriptores de la lista final.

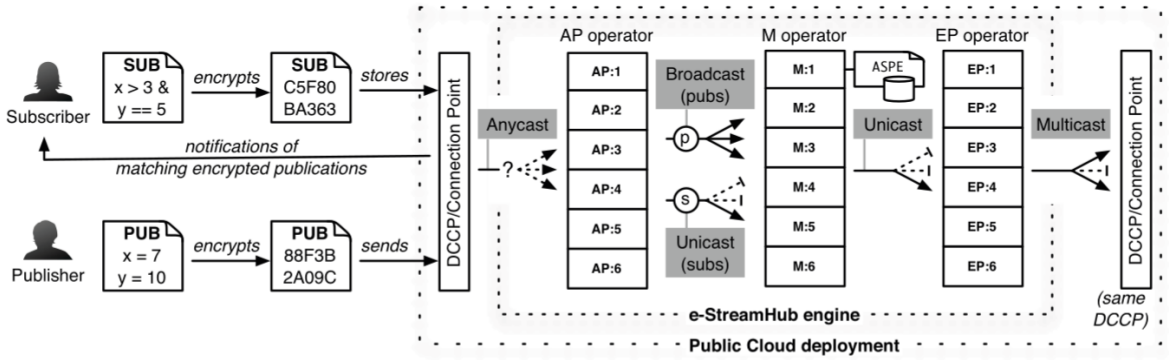

Figura 2.1: Ejemplo de un sistema STREAMHUB desplegado en la nube, con 6 instancias de cada operador. Los eventos van de izquierda a derecha. Imagen obtenida de [6].

La elasticidad de E-StreamHub está implementada mediante la creación de nuevos operadores en nuevos host cuando estos se necesiten, duplicando los mensajes a procesar en una memoria temporal, de forma que cuando el nuevo operador esté configurado, pueda comenzar a procesar estos mensajes, filtrando los obsoletos y minimizando el impacto de esta operación en el sistema.[6]

Para poder configurar y manejar los operadores, y las operaciones de creación y eliminación de estos, E-StreamHub implementa y utiliza un administrador, que monitoriza el estado de los operadores y lleva a cabo las operaciones de escalado (añadir o eliminar operadores), lo que permite uniformidad en el sistema y en el manejo de las operaciones de escalado, estableciendo reglas comunes. Al tener que aplicar una configuración uniforme a todos los operadores, y con el objetivo de poder aplicar esta configuración y tener control sobre los operadores existentes, E-StreamHub también hace uso de ZooKeeper[2].

Las reglas a seguir para aplicar las operaciones de escalado se basan en el estado

[2]https://zookeeper.apache.org/

de ciertos recursos de computación, como el uso de CPU, memoria, etc. Estas reglas, que se aplican para minimizar los costes a la vez que se maximiza la eficiencia del sistema, se dividen en *locales*, que provocan que se muevan operadores entre host ya existentes; y *globales*, que hacen que el sistema escale, añadiendo o quitando host y moviendo los operadores a otros host con poca carga de trabajo.

2.2. Problemas de disponibilidad de cargas de trabajo basadas en contenido

Uno de los principales problemas actuales en la investigación y desarrollo de sistemas publicador/subscriptor es la carencia de cargas de trabajo reales y públicas, que permitan probar los sistemas en desarrollo en unas condiciones lo más cercanas a las reales. Estas pruebas son de vital importancia para el desarrollo de este tipo de sistemas, ya que ofrecen datos y métricas sobre el comportamiento final del sistema.

Esta carencia es consecuencia del contenido de los propios datos presentes en las cargas de trabajo, ya que publicar estas cargas de trabajo puede suponer un problema de seguridad para los usuarios y la propia empresa que los publica, al contener los intereses (subscripciones) de los usuarios[14][15][8].

2.3. Auto-escalado

El auto-escalado es la capacidad de un sistema de ajustar (aumentar/disminuir) los recursos computacionales de os que hace uso de forma automática, en base a la cantidad de trabajo que está recibiendo, y con mínima o ninguna supervisión humana. Esta característica se ha popularizado en gran medida en los sistemas de computación en la nube, ya que minimiza los costes en un entorno en el que los recursos son *pay-as-you-go*, es decir, solo se paga por los recursos que se usan en cada momento.

Este tipo de sistemas siguen unos requisitos mínimos, especificados en el Service Level Agreement o *SLA*, de sus siglas en inglés, el cual es un contrato entre el cliente y el propietario de la aplicación. Estos requisitos, por lo general, son las métricas mínimas del sistema, como por ejemplo, el tiempo de respuesta, el número de mensajes por segundo producidos por el sistema (*throughput*[3]).

El escalado de un sistema puede llevarse a cabo de dos formas diferentes[15]:

- **Escalado horizontal** (*scaling in/out*): añadir o eliminar recursos mediante replicar los ya existentes con el objetivo de balancear el trabajo. Por ejemplo, crear o eliminar una instancia del host para que procese parte del trabajo.
- **Escalado vertical** (*scaling up/down*): añadir o eliminar recursos computacionales a un host que ya esté procesando el trabajo recibido. Por ejemplo, aumentar o disminuir la cantidad de CPU dedicada al procesamiento del trabajo recibido.

El escalado horizontal es ampliamente usado en los sistemas de computación en la nube, al ofrecer mayor flexibilidad a los usuarios en el uso de los recursos computacionales[4]. En cambio, el escalado vertical no se usa en gran medida, ya que, por lo

[3]Para mayor coherencia y brevedad, se usará *throughput* en el resto del documento
[4]En este trabajo solo se tratará el escalado horizontal.

general, los sistemas operativos no permiten cambiar la cantidad de recursos dedicados a una tarea mientras esta se está llevando a cabo sin reiniciar su procesado.

Adicionalmente, los sistemas auto-escalables se pueden clasificar en reactivos y predictivos, es decir, en base a su forma de reaccionar ante una situación que requiera de una operación de escalado.

2.3.1. Reactivo

Los sistemas auto-escalables que no realizan predicción alguna en la fase de análisis emplean el *auto-escalado reactivo*, es decir, se limitan a responder al estado actual del sistema, llevando a cabo las operaciones de escalado requeridas en ese momento, sin tener en cuenta el estado en el que se podrá encontrar el sistema en el futuro inmediato.

De esta forma, si el sistema se encuentra en una situación en la que la operación de escalado que se lleva a cabo no es suficiente para la carga de trabajo que está recibiendo, tendrá que realizar una nueva operación de escalado. Este procedimiento puede llevar al sistema, hasta que realice las suficientes operaciones de escalado, a proporcionar un servicio por debajo del óptimo y/o esperado.

2.3.2. Predictivo

A diferencia del auto-escalado reactivo, el auto-escalado predictivo sí tiene en cuenta el futuro inmediato del sistema mediante la aplicación de algoritmos de predicción basados en diferentes métricas, las cuales pueden basarse, por ejemplo, en el grado de utilización de cierto recurso del sistema en una ventana de tiempo previa a la realización de la operación de escalado, en las métricas propias del sistema, como tiempos de respuesta, *throughput*, etc.

Este tipo de escalado es capaz de adaptar los recursos disponibles de la forma más óptima del sistema a casi cualquier situación en la que aumente la carga de trabajo del sistema, a excepción de los incrementos repentinos y sin precedentes, ya que estos son completamente impredecibles.

2.4. Cargas de trabajo

Las cargas de trabajo utilizadas en los sistemas publicador/subscriptor representan el número de peticiones de usuarios al sistema, junto con su tiempo de envío[16].

Existen 5 tipos de cargas de trabajo en los sistemas de computación en la nube:

- *Carga estática*: carga con un número constante de eventos por minuto (ver Figura 2.2).
- *Carga creciente*: carga con un rápido incremento en los eventos por minuto (ver Figura 2.3).
- *Carga con crecimiento puntual*: carga que presenta un incremento puntual en los eventos por minuto.
- *Carga periódica*: carga con cambios periódicos en el número de eventos por minuto (ver Figura 2.4).

- *Carga "On-&-off"*: carga que presenta periodos en los que el número de eventos por minuto es 0 (ver Figura 2.4).
- *Carga impredecible*: carga que presenta fluctuaciones aleatorias en el número de eventos por minuto (ver Figura 2.5).

En este Trabajo de Fin de Grado, solo se harán uso de las cargas crecientes y con crecimiento puntual, al ajustarse a los tipos de pruebas que se desarrollarán, ya que estos permitirán, mediante los resultados obtenidos, poder diseñar e implementar los estudios y/o mejoras necesarias en el auto-escalado del sistema.

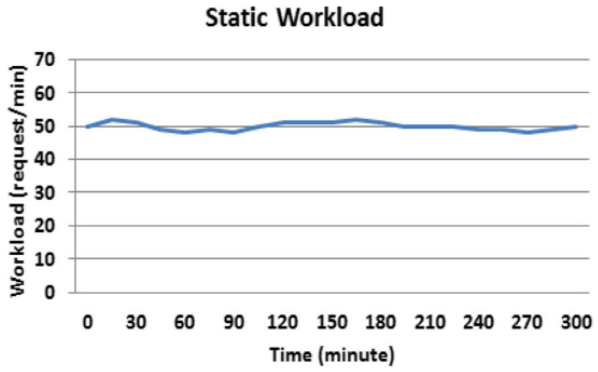

Figura 2.2: *Carga estática.* Imagen obtenida de [16].

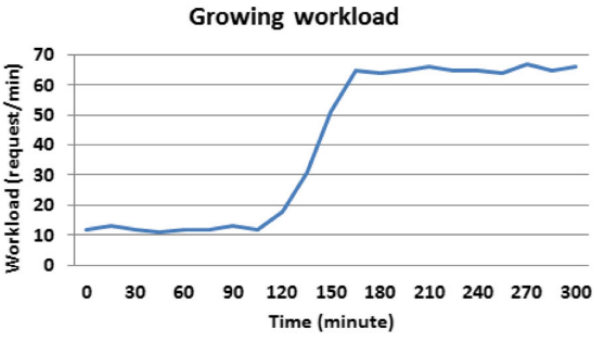

Figura 2.3: *Carga creciente.* Imagen obtenida de [16].

2.5. Estado del Arte

2.5.1. E-SilboPS

El sistema E-SilboPS, continuación natural del sistema SilboPS y el resultado del trabajo de una tesis doctoral[3], es un sistema publicador/subscriptor basado en contenido y en contexto, que ha sido diseñado y desarrollado de forma que sea elástico por defecto.

La arquitectura de este sistema, como se puede comprobar en la Figura 2.7 está compuesta por cuatro capas de operadores, similar a la encontrada en el sistema *StreamHub*[13], estando cada capa compuesta por una o varias instancias del correspondiente operador, siendo estos últimos los siguientes:

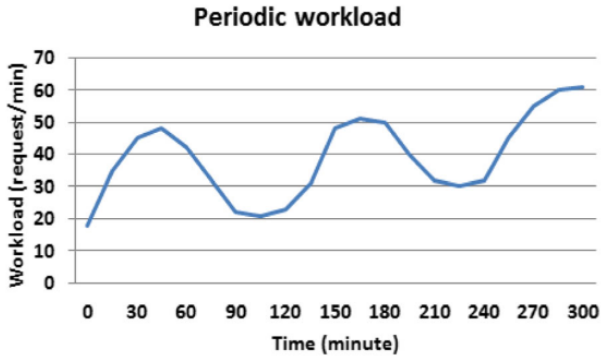

Figura 2.4: *Carga periódica.* Imagen obtenida de [16].

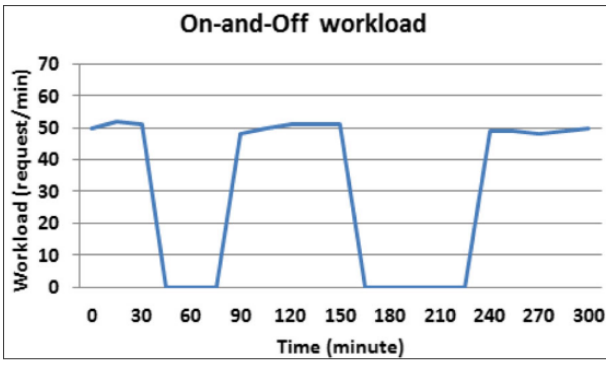

Figura 2.5: Carga *"On-&-off"*. Imagen obtenida de [16].

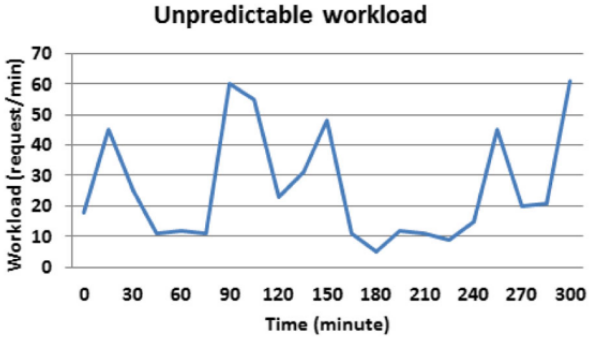

Figura 2.6: *Carga inpredecible.* Imagen obtenida de [16].

Connection Points (CP)
Estos operadores se encargan de la conexión con los diferentes publicadores y subscriptores y del envío de las publicaciones correspondientes a los subscriptores interesados en ellas. Los *CP*, que reciben tanto publicaciones como subscripciones, envían estas a un *Access Point* libre.

Access Points (AP)
Los *Access Points*, que reciben publicaciones y subscripciones por igual, se encargan de enviar la subscripción recibida al *Matcher* seleccionado mediante un selector; mientras que si reciben una publicación, envían una copia de esta a cada *Matcher*.

Matcher (M)
Los *Matchers* almacenan las subscripciones de los usuarios, teniendo cada instancia una pequeña lista de subscripciones, que serán las que le ha asignado el selector. En cambio, las publicaciones se usan para comparar los parámetros del contenido deseado por cada subscripción con los de la publicación manejada, generando una lista de subscriptores interesados en dicha publicación.

Exit Points (EP)
Estos operadores, que solo reciben las listas parciales de subscriptores generadas por los *Matchers* y la publicación correspondiente, componen una lista completa de subscriptores interesados en dicha publicación.

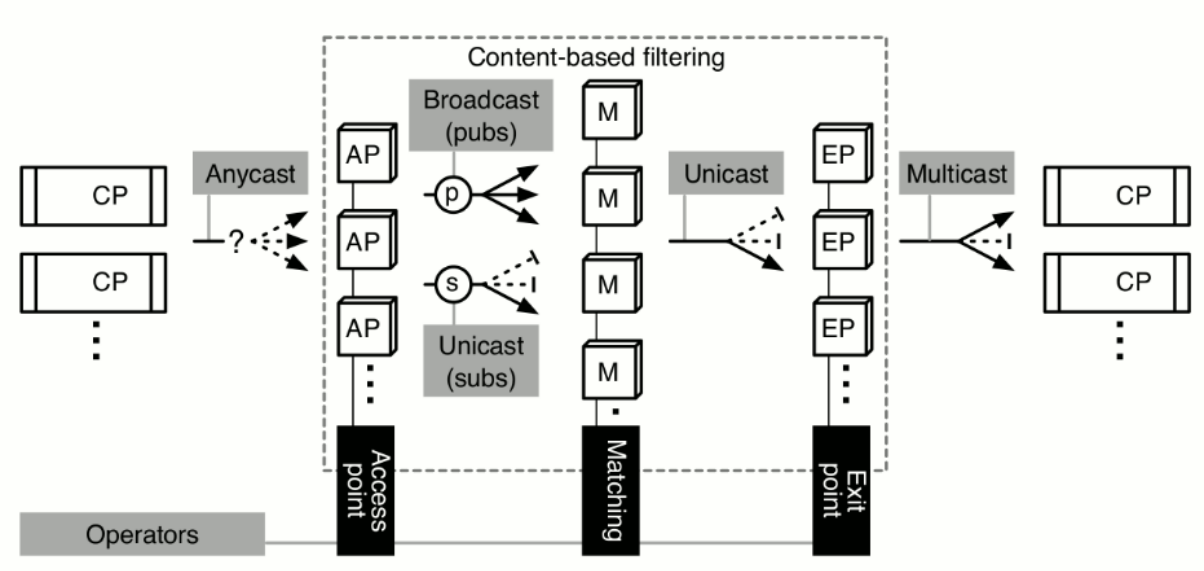

Figura 2.7: Arquitectura y funcionamiento de los sistemas *E-SilboPS* y *StreamHub*. Imagen obtenida de [6].

La elasticidad de *E-SilboPS* se basa en la capacidad de cada capa de operadores de añadir o quitar instancias (operaciones de escalado) de los operadores correspondientes de forma dinámica, ajustando la capacidad del sistema a los requisitos de este en cada momento, optimizando el uso de los recursos computacionales al máximo.

Esta escalabilidad dinámica sin interrumpir el sistema se lleva a cabo gracias al algoritmo de escalado, basado en el algoritmo de Chandy-Lamport[17], que permite

al sistema seguir operando de forma normal incluso cuando se está llevando a cabo una operación de escalado.

De forma adicional, y para poder mantener la coordinación y comunicación entre las diferentes capas de operadores del sistema, como se observa en la Figura 2.8 y la consistencia y estabilidad del mismo, se requiere del uso de un administrador, que imponga la configuración establecida para cada y su coordinación con las demás. El sistema elegido para llevar a cabo esta tarea es Zookeeper[5].

La configuración del sistema y el número de operadores en cada capa se expresa mediante el formato *X-Y-Z*, siendo este el número de instancias de los operadores *CP/AP*, *M* y *EP*, respectivamente. A lo largo de este documento se usará este formato para expresar la configuración del sistema.

2.6. Herramientas utilizadas

Java

Java[6] es un lenguaje de programación orientado a objetos y que sigue un paradigma imperativo. Ver logo en Figura 2.9.

Tanto E-SilboPS, como los proyectos anteriores se han implementado en Java, por lo que este trabajo utiliza este lenguaje para desarrollar las pruebas necesarias para medir el rendimiento del sistema.

R

R[7] es un lenguaje de programación orientado al análisis estadístico. Ver logo en Figura 2.10.

Se ha utilizado R para la realización de las cargas de trabajo, y la generación de las gráficas incluidas en este TFG, así como la implementación de los modelos predictivos, debido a la versatilidad y cantidad de herramientas que proporciona para llevar a cabo estas tareas.

Git

Git[8] es una herramienta abierta y de código libre para el control de versiones diseñada para facilitar el manejo de proyectos, principalmente de código. Ver logo en Figura 2.11.

Para mantener coherencia en el desarrollo y tener un control de versiones del código, se ha utilizado Git, además de GitHub[9] y GitLab[10], herramientas web necesarias para facilitar el manejo y gestión de las diferentes ramas y de los cambios realizados.

[5]https://zookeeper.apache.org/
[6]https://www.java.com/es/
[7]https://www.r-project.org/
[8]https://git-scm.com/
[9]https://github.com/
[10]https://github.com/

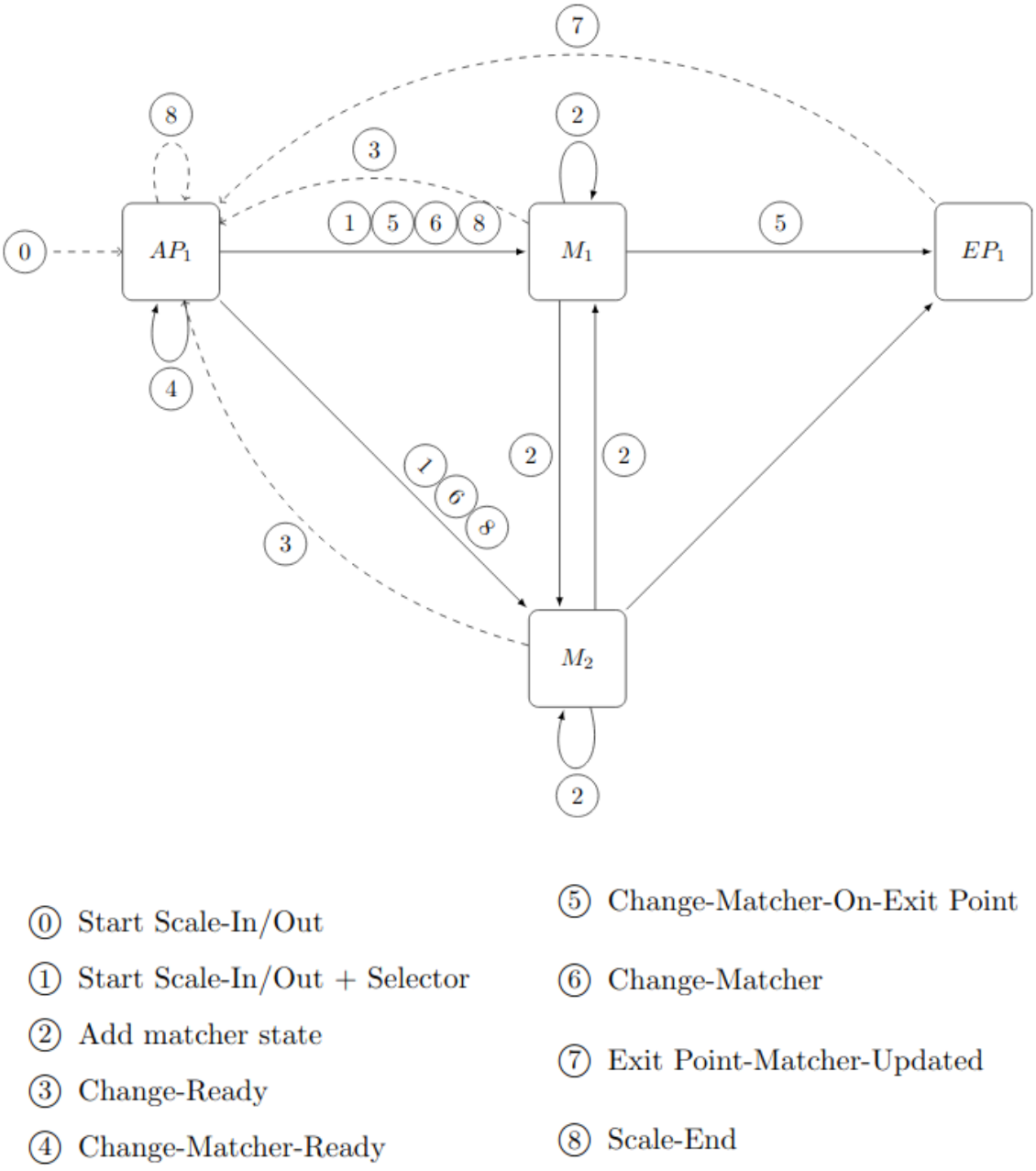

Figura 2.8: Comunicación y coordinación de las capas de operadores en una operación de escalado de la capa de *Matchers* (M2 es la nueva instancia de *Matcher*). Las lineas continuas representan la comunicación de subscripciones y publicaciones entre capas de operadores, mientras que las discontinuas son mensajes de coordinación para sincronizar las capas y los operadores. Imagen obtenida de [2].

Figura 2.9: Logo de Java.

Figura 2.10: Logo de R.

Figura 2.11: Logo de Git.

Apache Maven

Maven[11] es una herramienta para el gestión y creación de proyectos Java, ofreciendo un modelo de configuración muy accesible y sencillo de manejar para el desarrollador. Ver logo en Figura 2.12.

Se ha utilizado Apache Maven debido a su uso previo en este proyecto, así como la facilidad que proporciona a los desarrolladores para gestionar proyectos Java.

Figura 2.12: Logo de Apache Maven.

JUnit 5

JUnit 5[12] es un framework para el desarrollo de pruebas para aplicaciones Java que ha sido ampliamente usado. Ver logo en Figura 2.13.

Para implementar las pruebas del generador de cargas de trabajo reales, así como las pruebas propias el sistema E-SilboPS, se han usado pruebas de JUnit 5. Este framework permite validar de forma muy completa el funcionamiento de una aplicación Java, pudiendo probar y confirmar que el comportamiento y las funciones de un sistema son las esperadas.

Figura 2.13: Logo de JUnit 5.

[11] https://maven.apache.org/index.html
[12] https://junit.org/junit5/

Capítulo 3

Desarrollo de un generador de cargas de trabajo para sistemas publish/subscribe basados en contenido

En esta sección se expone el trabajo realizado durante el desarrollo de este generador, así como su contexto y motivación, y los resultados obtenidos.

3.1. Motivación

La necesidad de este generador se origina en la dificultad de probar el sistema *E-SilboPS* con cargas de trabajo con datos reales (2.2), habiendo sido probado solo con cargas de trabajo sintéticas (artificiales), es decir, generadas por un programa en base a ciertos parámetros estadísticos (número de subscripciones totales, número de subscripciones por segundo, número de notificaciones, número de notificaciones por segundo, etc).

El principal problema con este tipo de pruebas es que no reflejan el comportamiento que tendrá el sistema en un entorno real, incluso cuando las cargas generadas intentan reflejar dicho entorno. Para solventar esto, se necesita una carga de trabajo real compatible con este sistema.

Para llevar a cabo estas pruebas con la carga de trabajo real, se necesita generar, a partir de esta, una carga compatible con *E-SilboPS*, transformando cada evento al formato aceptado por este sistema, preservando el orden y contexto original.

3.2. Carga de trabajo

Esta carga de trabajo se ha extraído del repositorio de *datasets* del *Middleware Systems Research Group*[1] de la Universidad de Toronto, y pertenece al proyecto *Mammoth*[2], que se ha creado por medio de obtener la información de los eventos de un

[1] http://msrg.org/

[2] http://msrg.org/datasets/mammoth

juego *MMO*[3] que funciona mediante un sistema *publish-subscribe* basado en contenido.

Además de haberse generado a partir de un sistema similar al que se ha usado en este proyecto, es lo suficientemente amplia para llevar a cabo las pruebas necesarias en nuestro sistema, y presenta un elevado número de subscripciones y de-subscripciones, cualidad necesaria para comprobar las fluctuaciones causadas por los incrementos y decrementos de la cantidad de trabajo recibida.

3.2.1. Análisis estático

Previo uso de la carga, se ha llevado a cabo un análisis estático, con el objetivo de comprobar su compatibilidad con el formato de los eventos de *E-SilboPS* y la integridad de los propios datos, y, además, poder conocer de antemano datos estadísticos relevantes para el futuro desarrollo y especificación de pruebas que la usen.

Mediante este análisis, se ha obtenido la siguiente información básica sobre la carga de trabajo:

- La marca de tiempo (*timestamp*)[4], está compuesta por `13` dígitos, a diferencia de la de *Linux*, de `10` dígitos.
- Contiene mensajes de error (`[ERROR] ...`), que han de ser eliminados.
- Tiene mensajes generados de manera simultánea, ya que el *timestamp* de algunos mensajes es igual.

De forma adicional, se ha generado la siguiente información estadística, útil para el desarrollo futuro de este proyecto:

- Número de subscripciones: `113.904`
- Número de de-subscripciones: `112.682`
- Número de publicaciones: `6.207.207`

Con el objetivo de comprobar la correctitud de los eventos de la carga de trabajo, se ha verificado que la carga no contiene subscripciones y de-subscripciones duplicadas, o de-subscripciones antes que la subscripción correspondiente.

3.2.2. Formato y operadores

Para concluir el análisis estático, se han comparado los formatos de cada tipo de eventos que, de acuerdo a la documentación de Mammoth[18] y E-SilboPS[2], ambos sistemas cuentan con los siguientes operadores para los pares clave-valor de sus eventos.

Como se observa en el Tabla 3.1, estos sistemas solo comparten el operador de igualdad (`eq`), por lo que la carga de trabajo generada solo contendrá este operador para todas las pares clave-valor de cada uno de los eventos presentes en la misma.

El formato de los eventos en ambos sistemas son también diferentes entre sí para los tres tipos de eventos que manejan, siendo el formato de las subscripciones, como se

[3] *Massive Multiplayer Online Game* - Videojuego online con multitud de jugadores a través de Internet
[4] En este trabajo, el término *timestamps* y *marca de tiempo* se usarán de forma intercambiable

E-SilboPS	Mammoth	Descripción
{"exists": ...}	-	El atributo existe
{"=": ...}	eq	Igual
{"!=": ...}	-	No igual
{">": ...}	-	Mayor que
{">=": ...}	-	Mayor o igual que
{"<": ...}	-	Menor que
{"<=": ...}	-	Menor o igual que
{"^": ...}	-	Comienza con el prefijo
{"$": ...}	-	Termina con el sufijo
{"contains": ...}	-	Contiene

Cuadro 3.1: Tabal comparativa de operadores de *E-SilboPS* y *Mammoth*

ve en los Listing 3.1 y Listing 3.2, compatible, pues los *"constraints"* son las claves y valores que se comprueban contra las publicaciones, que en *Mammoth* son los valores entre corchetes, mientras que el valor entre llaves es el *timestamp* de la subscripción, la cual solo se usará para mantener el orden de los eventos en el fichero.

```
{
        "id" : "",
        "contextFunction": {},
        "constraints": {
                "symbol:str": [{"=": "GOOG"}],
                "value:double": [{">": 100.25}, {"<=": 500.0}],
                "volume:long": [{">=": 1000}],
                "timestamp:long": [{"exists": ""}]
        }
}
```

Listing 3.1: Ejemplo de subscripción de E-SilboPS.

```
{1307423528250} s [class,eq,broadcast] (rmi://10.0.1.2:1099/Broker2-M4894)
```

Listing 3.2: Ejemplo de subscripción de Mammoth.

A diferencia que en *Mammoth*, como se ve en Listing 3.3 y Listing 3.4, el formato de las de-subscripciones en *E-SilboPS* es igual al de las subscripciones. A pesar de esto, la compatibilidad se mantiene.

```
{
        "id" : "",
        "contextFunction": {},
        "constraints": {
                "symbol:str": [{"=": "GOOG"}],
                "value:double": [{">": 100.25}, {"<=": 500.0}],
                "volume:long": [{">=": 1000}],
                "timestamp:long": [{"exists": ""}]
        }
}
```

Listing 3.3: Ejemplo de de-subscripcion de E-SilboPS.

```
{1307425244397} us rmi://10.0.1.4:1099/Broker4-M81393
```

Listing 3.4: Ejemplo de de-subscripción de Mammoth.

En las publicaciones, el formato de ambos sistemas son muy similares y compatibles, como se puede ver en Listing 3.5 y Listing 3.6.

```
{
        "symbol:str" : "GOOG",
        "value:double" : 333.2,
        "volume:long" : 345
}
```

Listing 3.5: Ejemplo de publicación de E-SilboPS.

```
{1307422272885} p [class,ProxyID_-3884500000][x,0.0][y,0.0]
```

Listing 3.6: Ejemplo de publicación de Mammoth.

3.3. Diseño

El programa deberá leer cada línea del fichero de entrada, es decir, la carga de trabajo proveniente de *Mammoth*, interpretar cada evento y generar el mismo evento en el formato compatible con *E-SilboPS*, manteniendo los datos y el orden de los mismos. El fichero generado debe ser en formato *JSON*[5], ya que es ese el formato de los eventos de *E-SilboPS*.

En este sistema, al tener las subscripciones y las de-subscripciones el mismo formato, el generador tiene que ser capaz de guardar todas las subscripciones que haya generado, y generar las de-subscripciones de las mismas cuando así lo lea del fichero de entrada.

De igual forma, ha de ser capaz de detectar errores en los eventos leídos, comprobando la correcta estructura de cada regla, sus valores, y sus operadores.

3.4. Implementación

El generador se ha implementado en el lenguaje de programación *Java*[6], utilizando las herramientas, *Maven*[7] para gestionar el proyecto y sus dependencias, y *JUnit5*[8] para realizar pruebas y comprobar el correcto funcionamiento del programa.

A partir del diseño propuesto, se ha creado y seguido la estructura de clases que se ve en la Figura 3.1, estableciendo una estructura de funciones transparente al usuario.

Las clases y sus funciones mostradas en la figura Figura 3.1 se han simplificado para facilitar su lectura y comprensión general, ya que no se han incluido estructuras de datos internas y los tipos de las mismas por ser únicamente de uso interno.

[5] JavaScript Object Notation
[6] https://www.java.com/en/
[7] https://maven.apache.org/
[8] https://junit.org/junit5/

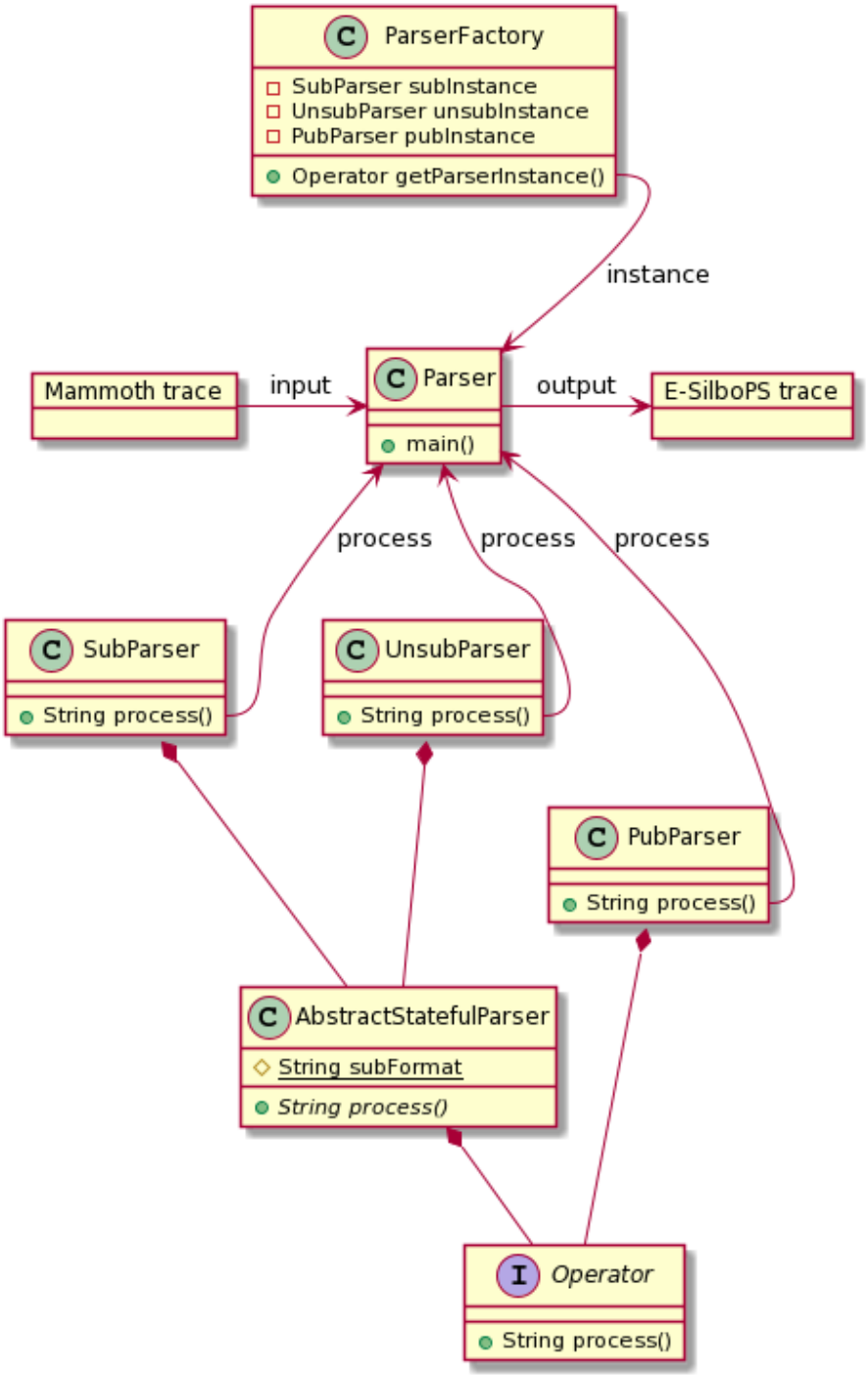

Figura 3.1: Diagrama de clases del generador de cargas de trabajo reales.

Para facilitar la lectura del fichero *JSON* de salida, este se ha implementado como una lista, o *array*, de objetos *JSON*, de forma que la prueba que lea dicho fichero solo tenga que leer cada uno de estos elementos en la lista para poder usarlos, sin necesidad de realizar más operaciones, las cuales serían necesarias si el fichero de salida no siguiese este formato.

Una vez implementado el generador, y habiendo generado el fichero *JSON* que representa la carga de trabajo real en el formato de *E-SilboPS*, se puede comenzar el desarrollo de las pruebas necesarias en este sistema usando esta carga.

3.5. Pruebas en E-SilboPS

Una vez generada la carga de trabajo deseada, se han creado las siguientes pruebas que la usan, con el objetivo de comprobar el comportamiento del sistema por medio de sus principales métricas de rendimiento, de forma que se obtenga una visión general del sistema a lo largo del tiempo.

Como no todas las publicaciones generan subscripciones, es necesario introducir dos nuevos eventos (una subscripción al inicio (ver Listing 3.7), y una publicación al final (ver Listing 3.8)), a la carga, que solo se enviará una única vez, marcando el final de la prueba correspondiente.

```
{
        "id" : "MAMMOTH_FLAG"
        "contextFunction" : {},
        "constraints" : {
                "class:str" : [{"=" : "END"}]
        }
}
```

Listing 3.7: Subscripción en E-SilboPS para identificar el final de la carga.

```
{
    "id:long" : 0
    "class:str" : "END"
}
```

Listing 3.8: Publicación en E-SilboPS que marca el final de la carga.

Una vez diseñados e implementados estos conceptos, se han implementado las siguientes pruebas, usando una topología 1-1-1 (1 EP, 1 M, 1 EP) en E-SilboPS, siendo el emisor y el receptor diferentes hilos (*threads*) de un mismo computador.

3.5.1. Velocidad fija de envío

Para esta primera prueba, se ha procedido al envío de toda la carga de trabajo, subscripciones, de-subscripciones y publicaciones, a diferentes velocidades de envío (ver lista a continuación) mediante repetir esta prueba con cada una de ellas.

- 50.000 mensajes por segundo.
- 100.000 mensajes por segundo.
- 200.000 mensajes por segundo.
- 500.000 mensajes por segundo.
- 1.000.000 mensajes por segundo.
- 6.433.794 mensajes por segundo (máxima velocidad posible).

Parámetros adicionales de estas pruebas:

- Número de publicaciones: 6.207.207
- Número de subscripciones: 113.904
- Número de de-subscripciones: 112.682
- Número de notificaciones: 4.515.781 (72 % de las publicaciones machean)
- Cada publicación machea como máximo, con una única subscripción.[9]

Para poder obtener los datos más precisos, se ha decidido medir el tiempo de respuesta una vez cada 50.000 notificaciones recibidas[10], ya que medir más frecuentemente provoca una distorsión en las mediciones y por ende, en su interpretación.

3.5.2. Secuencia logarítmica de subscripciones

En esta prueba, se separan las subscripciones y las publicaciones, enviando las subscripciones siguiendo una secuencia logarítmica (comenzando en 100 subscripciones), hasta haber enviado todas las subscripciones (113.904). Este proceso se muestra en la Figura 3.2). Tras enviar las nuevas subscripciones, se envían todas las publicaciones de la carga, midiendo el throughput total del sistema.

Parámetros de la prueba:

- Número de publicaciones: 6.207.207
- Subscripciones en secuencia logarítmica, de 100 a 113.904 (todas las subscripciones)
- La prueba se repetirá 10 veces por cada punto en la gráfica, siendo el valor final de este la media del throughput total de cada repetición.
- Velocidad de envío (input rate): 100.000 mensajes por segundo.

3.5.3. Carga de subscripciones creciente

Para llevar a cabo esta prueba, primero se ha generado una traza de números (ver Figura 3.3) con un incremento constante hasta llegar a un número deseado, y manteniendo dicho número hasta el final de la misma.

Esta traza de números (Figura 3.3) representa el número de subscripciones presentes en cada momento (ver Sección 2.4), con cierto ruido/fluctuaciones, simulando un entorno real con nuevas subscripciones y de-subscripciones cada segundo.

Parámetros de la prueba:

- Subscripciones base: 10.000
- Subscripciones finales: 20.000
- Ruido: $\pm$ 2000 subscripciones
- Tiempo de la prueba: 600 segundos

[9] Esto se tendrá en cuenta para el desarrollo de futuras pruebas.
[10] Esta característica se ha mantenido en todas las pruebas realizadas

Figura 3.2: Diagrama de ejecución de prueba con subscripciones en secuencia logarítmica.

- Velocidad de envío (input rate): 50.000 mensajes por segundo

Esta traza es leída por la prueba, y, en base al número presente de subscripciones en el sistema, genera nuevas subscripciones o de-subscripciones, de forma que, cada segundo, el sistema contenga el número de subscripciones especificadas en esta, siguiendo el flujo de ejecución que se muestra en la Figura 3.4).

El envío de publicaciones se realiza a un ratio fijo por segundo (input rate), y de forma paralela al envío de subscripciones/de-subscripciones, interpretando las publicaciones como una lista circular hasta que se haya recorrido toda la traza de subscripciones. Una vez esto sucede, se envían las restantes publicaciones y finaliza la prueba.

3.5.4. Carga de subscripciones estática

Estas pruebas se han basado en la confección de una carga de trabajo basada en la real (ver Sección 2.4, *Carga estática*), en la cual la cantidad de subscriptores no cambia durante toda la ejecución de dicha prueba, es decir, se envía un número determinado de subscripciones al inicio de la prueba, y se mantienen esas subscripciones hasta acabar esta.

Junto con estas carga, se debe de especificar la velocidad a la que el sistema envía las publicaciones (input rate), para simular un entorno en el que el sistema recibe muchos eventos por segundo, que han de ser procesados y enviados, con el objetivo de ver el punto en el que el sistema satura parcialmente.

En estas pruebas, el ratio al que se envían las publicaciones será mayor que en las siguientes pruebas, ya que el sistema no está procesando nuevas subscripciones o

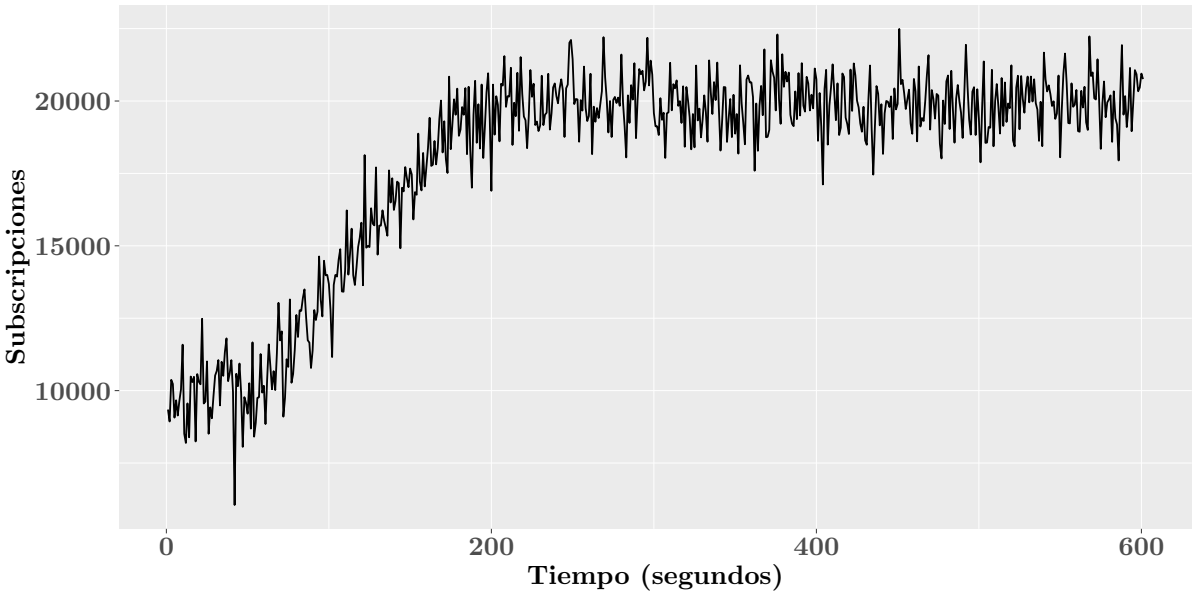

Figura 3.3: Ejemplo de carga de trabajo creciente (con ruido añadido).

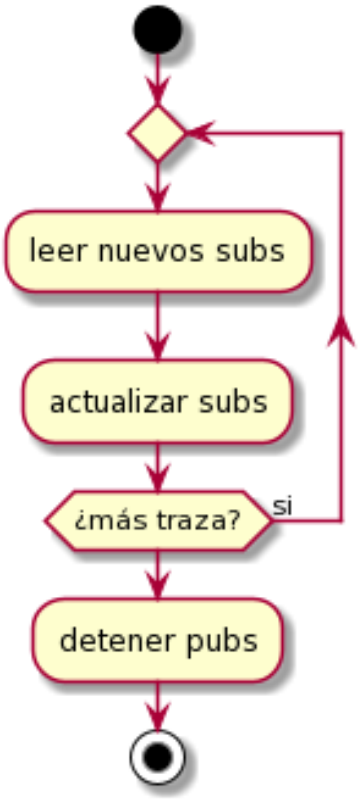

Figura 3.4: Diagrama de ejecución de la prueba con carga de trabajo incremental.

desubscripciones, solo publicaciones entrantes.

Parámetros de las pruebas:

- Dos pruebas, una con 50.000 subscripciones, otra con 100.000 subscripciones
- Tiempo de la prueba: 600 segundos
- Velocidad de envío (input rate): 200.000 mensajes por segundo

3.5.5. Carga de subscripciones con crecimiento puntual

Siguiendo con estas pruebas, se han desarrollado unas cargas de trabajo que presentan un incremento de las subscripciones puntual, es decir, las subscripciones comienzan constantes, y tras un periodo de tiempo, incrementan hasta cierto valor. Una vez se llega a ese valor, todas esas nueva subscripciones se eliminan mediante el envío de las desubscripciones correspondientes.

Subscripciones de las pruebas:

- Crecimiento de 20.000 a 100.000 subscripciones (Figura 3.5)
- Crecimiento de 25.000 a 113.904 subscripciones (Figura 3.6)
- Crecimiento rápido de 25.000 a 113.904 subscripciones (Figura 3.7)

Parámetros de las pruebas:

- Publicaciones: buffer circular de 6.207.207 publicaciones
- Velocidad de envío (input rate): 125.000 mensajes por segundo

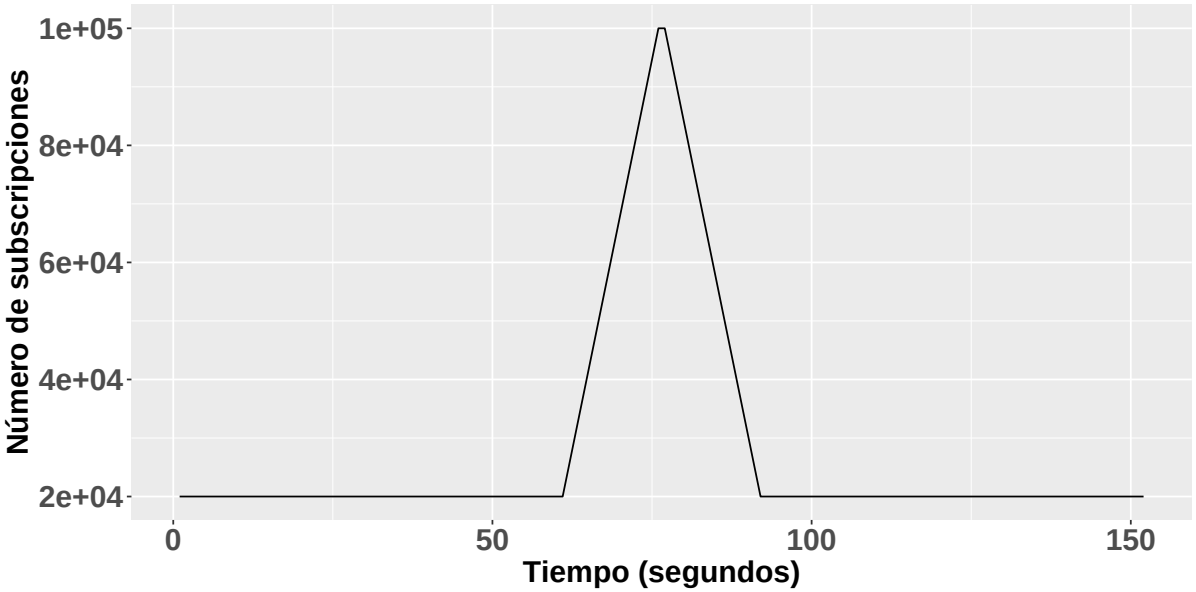

Figura 3.5: Carga base de 20.000 subscripciones con incremento hasta 100.000 subscripciones.

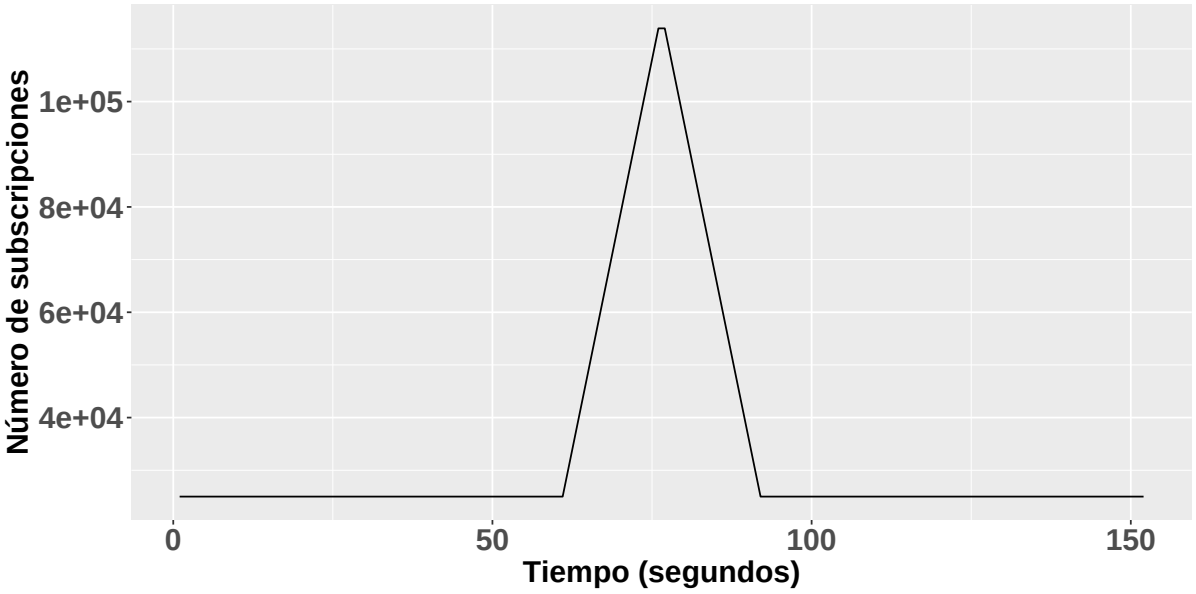

Figura 3.6: Carga base de 25.000 subscripciones con incremento hasta 113.904 subscripciones.

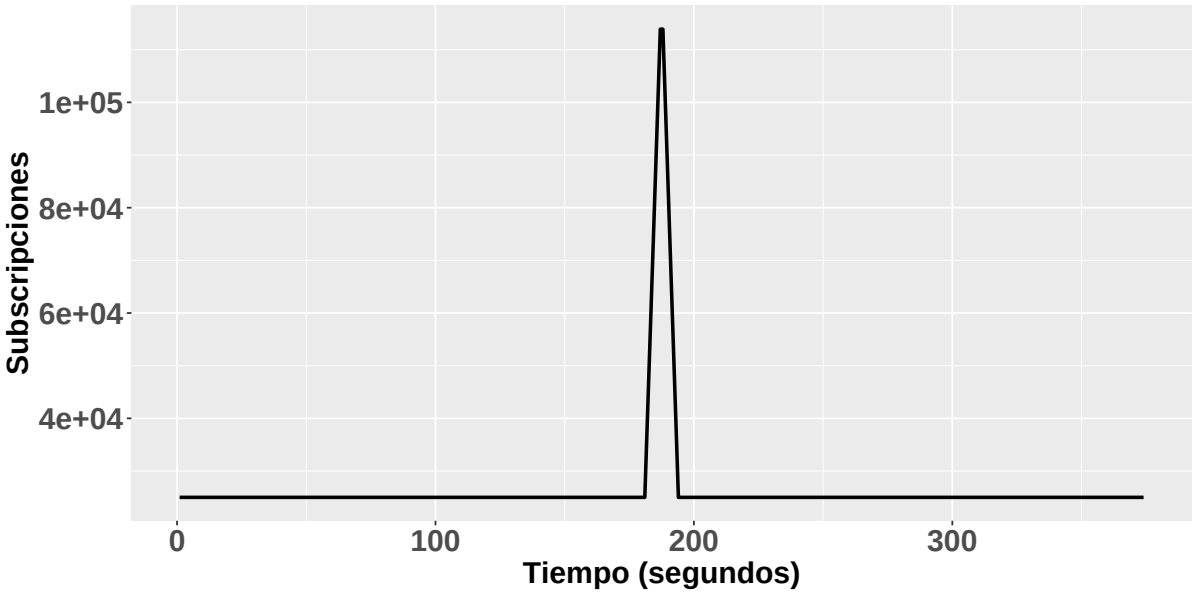

Figura 3.7: Carga base de 25.000 subscripciones con incremento muy rápido hasta 113.904 subscripciones.

3.6. Métricas de rendimiento

Para poder cuantificar y analizar el rendimiento del sistema, como se ha mencionado previamente, es necesario obtener las principales métricas de rendimiento, con el objetivo de identificar posibles problemas o ineficiencias en el sistema.

Estas métricas son:

- **Tiempo de respuesta** (*response time*): lapso de tiempo desde que se envía la publicación hasta que es recibida por un usuario interesado en ella.
- **Notificaciones enviadas a usuarios por segundo** (*throughput*): número de notificaciones, por segundo, que han salido del sistema hacia usuarios interesados en ellas

Para obtener estos datos, las pruebas implementadas en la Sección 3.5 han recogido los tiempos de envío y recepción de cada mensaje, mediante los cuales se han calculado los tiempos de respuesta de cada publicación (que ha llegado a algún subscriptor), y el número de mensajes totales del sistema, plasmando esta información en ficheros con el formato *CSV*[11] para poder dibujar gráficas y obtener conclusiones de las pruebas y del sistema.

3.7. Análisis de los resultados

Con los datos obtenidos por medio de las pruebas implementadas y ejecutadas en la Sección 3.5, se han obtenido los siguientes datos y gráficas[12].

Velocidad fija de envío

Esta prueba, como se ha mencionado en la Subsección 3.5.1, se ha llevado a cabo múltiples veces a diferentes velocidades de envío (input rate), para poder ver el comportamiento del sistema, tanto en throughput como en tiempo de respuesta de notificación.

- 50.000 mensajes por segundo

Los resultados de esta prueba, tanto del throughput (ver Figura 3.8) como del tiempo de respuesta (ver Figura 3.9) muestran un sistema que funciona sin saturar, pues el tiempo de respuesta es muy cercano a 0. El throughput muestra ciertos picos precedidos de valles, los cuales se deben al propio contenido de la carga de trabajo ya que hay zonas que provocan más subscripciones que otras, y no llega a las 50.000 notificaciones por segundo (salvo en los ya mencionados picos), debido a que no todas las publicaciones producen alguna notificación.

Conclusiones de la prueba

- Tiempo de respuesta mínimo y estable
- Throughput por debajo del input rate (esperable), picos debidos a la propia carga.

[11] Comma-Separated Values

[12] Ejes Y en escala logarítmica para mejor representación de los datos.

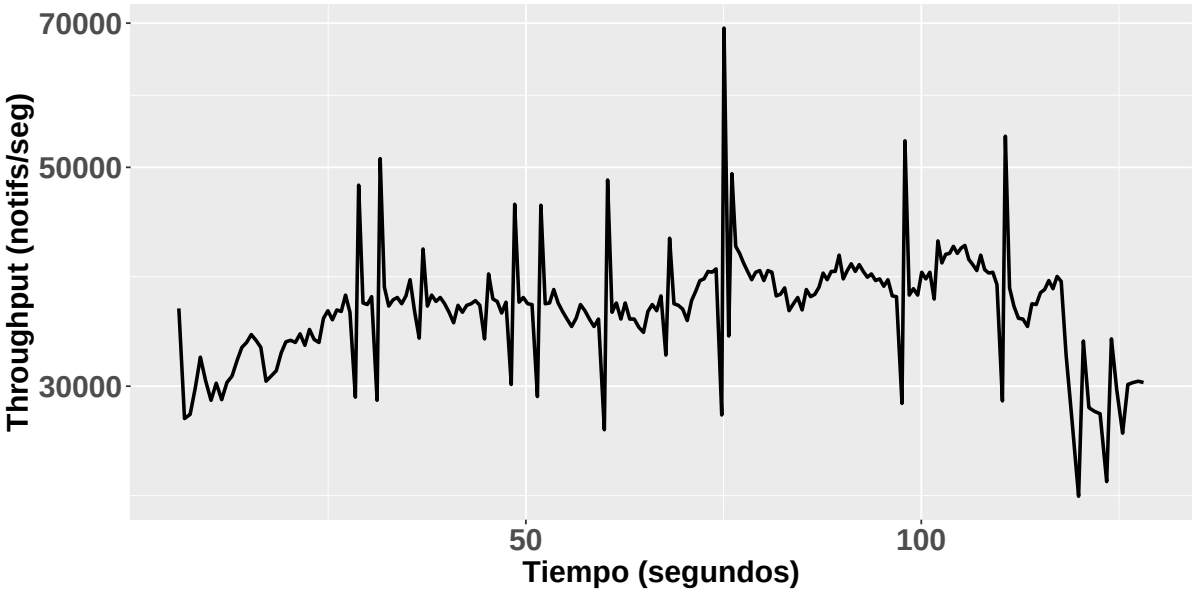

Figura 3.8: Throughput de prueba con carga completa e input rate de 50.000 mensajes/s.

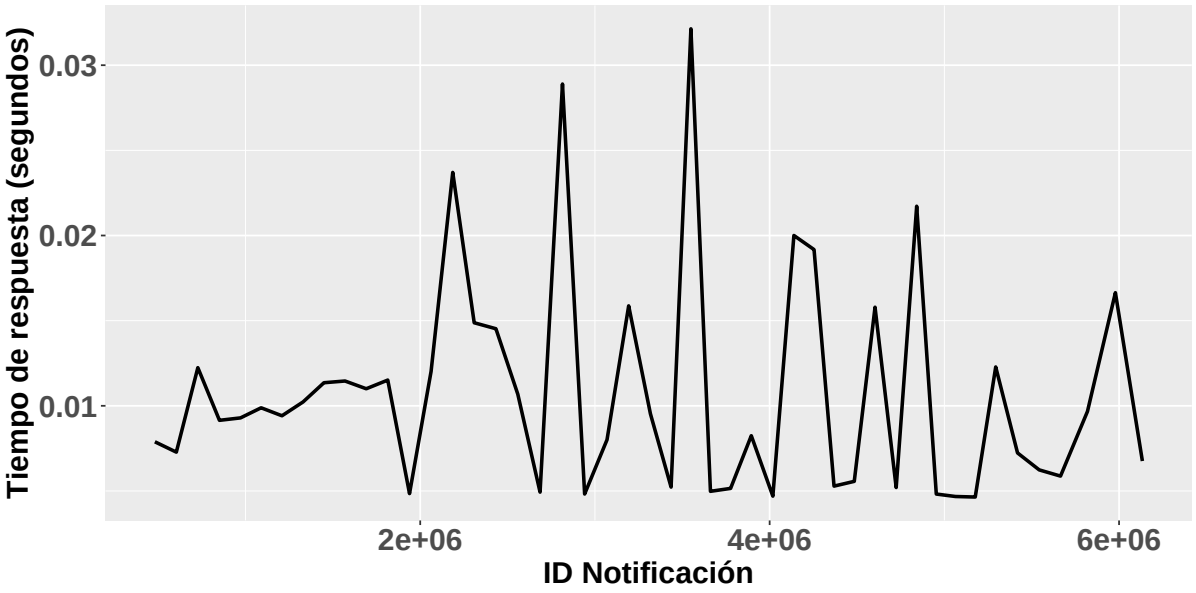

Figura 3.9: Tiempo de respuesta de prueba con carga completa e input rate de 50.000 mensajes/s.

- 100.000 mensajes por segundo

Los resultados de esta prueba, tanto en el throughput (ver Figura 3.10) como en el tiempo de respuesta (ver Figura 3.11), muestran que el sistema ha saturado de forma parcial, ya que el throughput cae muy por debajo del valor esperado, y el tiempo de respuesta crece de forma constante, lo que indica que el sistema no está procesando las publicaciones a tiempo, y se están encolando.

Además de esto, se puede ver claramente la relación entre ambas mediciones, ya que la caída del throughput (Figura 3.10, a los 50 segundos), concuerda con el incremento en el tiempo de respuesta (Figura 3.11 a partir de la notificación con ID 3.000.000).

Conclusiones de la prueba:

- El tiempo de respuesta crece de forma constante, ya que se encolan las notificaciones.
- El sistema satura de forma parcial, debido al throughput inestable, llegando a caer hasta los 3.000 mensajes por segundo.

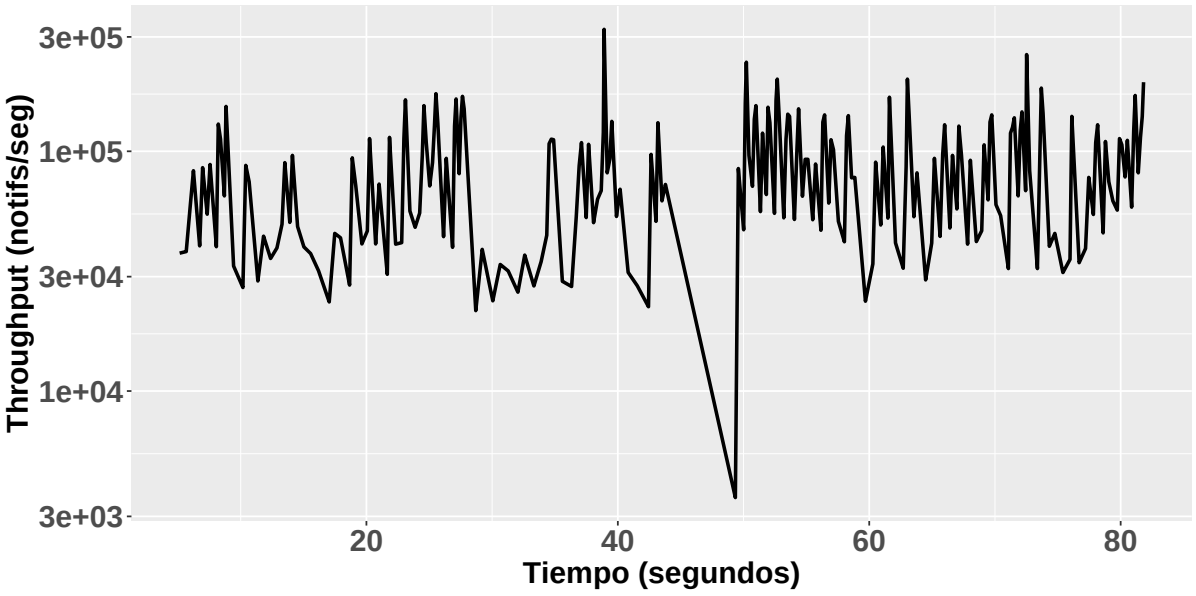

Figura 3.10: Throughput de prueba con carga completa e input rate de 100.000 mensajes/s.

- 200.000 mensajes por segundo

En esta prueba, los resultados del throughput (ver Figura 3.12) y del tiempo de respuesta (ver Figura 3.13, demuestran que el sistema ha saturado, pero su throughput no ha caído tanto como anteriormente, a diferencia del tiempo de respuesta, que sí ha crecido en gran medida, lo cual indica que las notificaciones se están encolando en gran medida, ya que el sistema no puede procesarlas a tiempo, llegando estas a llegar al cliente tras más de 40 segundos.

Conclusiones de la prueba:

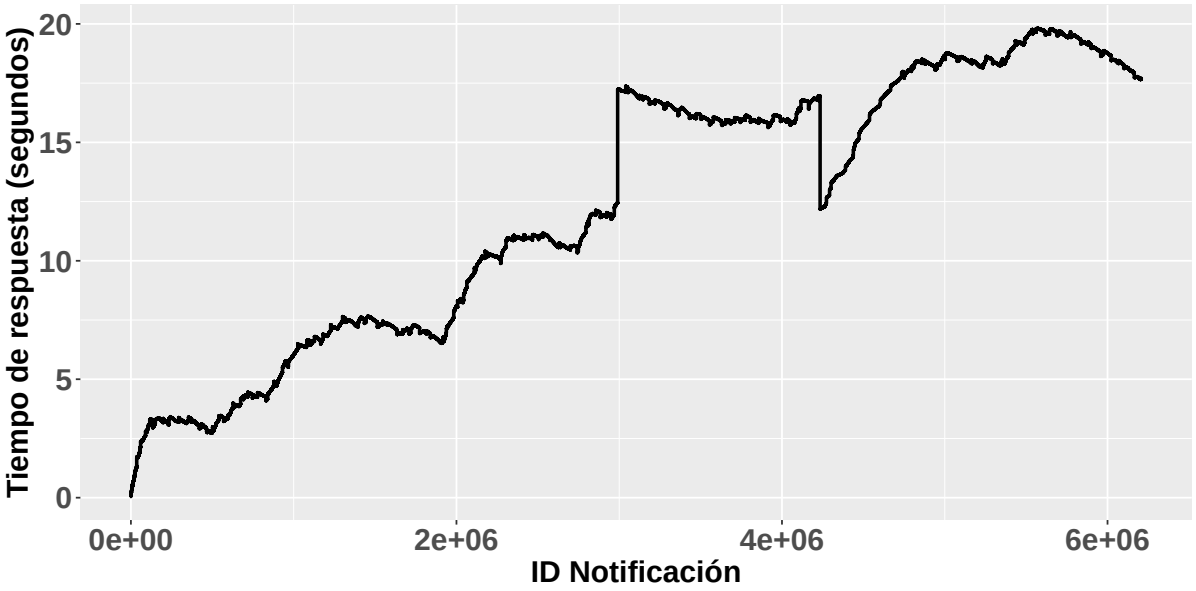

Figura 3.11: Tiempo de respuesta de prueba con carga completa e input rate de 100.000 mensajes/s.

- El tiempo de respuesta se incrementa de forma exponencial.
- El throughput presenta muchos picos al encolar muchas notificaciones que se procesan de forma muy rápida, llegando algunos picos a superar el input rate.

- 500.000 mensajes por segundo

Al igual que en las pruebas anteriores, se puede ver la saturación del sistema, tanto en el throughput (ver Figura 3.14) como en el tiempo de respuesta (ver Figura 3.15), presentando en el primero importantes caídas (hasta casi 3.000 de throughput), y un tiempo de respuesta que llega a casi 70 segundos.

En esta prueba también se puede apreciar la relación del throughput y del tiempo de respuesta, pues este último presenta dos repentinos incrementos, los cuales coinciden con severas caídas del throughput.

Conclusiones de la prueba:

- El tiempo de respuesta es mucho mayor que en anteriores pruebas, señal del grado de saturación del sistema.
- El throughput muestra severas caídas, y oscila en torno a los 100.000 mensajes por segundo, muy por debajo de los 500.000 de input rate.

- 1.000.000 mensajes por segundo

Los resultados de esta prueba son muy similares a los de la prueba anterior, con un throughput (ver Figura 3.16) con muchos picos alrededor de 75.000 mensajes por segundo, y que presenta 3 caídas severas, las cuales coinciden con 3 incrementos presentes en el tiempo de respuesta (ver Figura 3.17), el cual presenta mayores

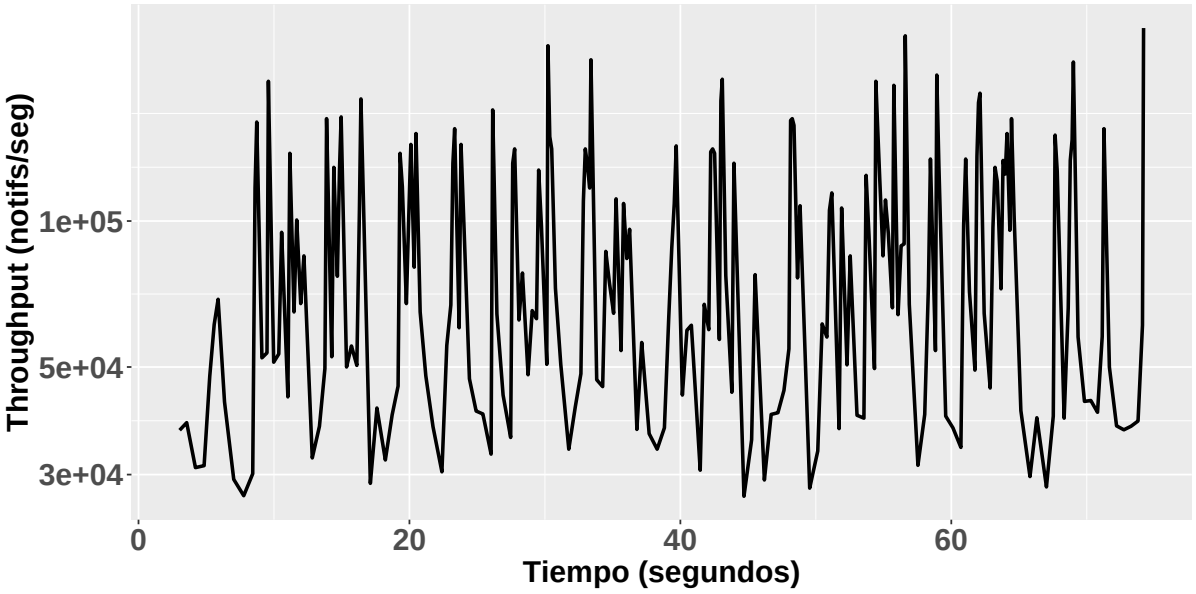

Figura 3.12: Throughput de prueba con carga completa e input rate de 200.000 mensajes/s.

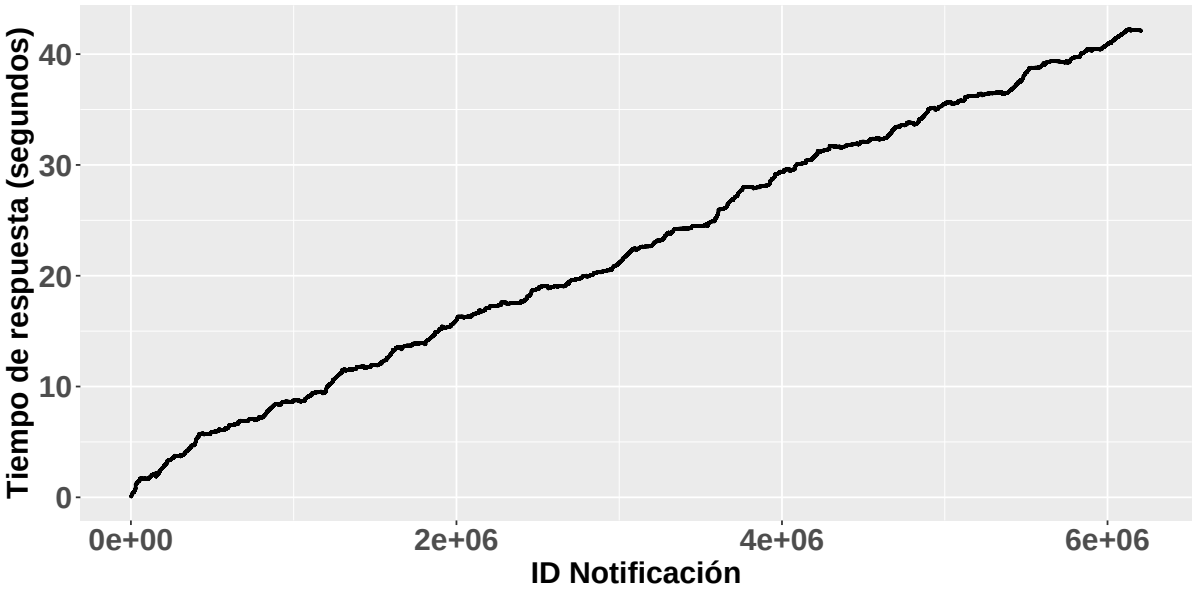

Figura 3.13: Tiempo de respuesta de prueba con carga completa e input rate de 200.000 mensajes/s.

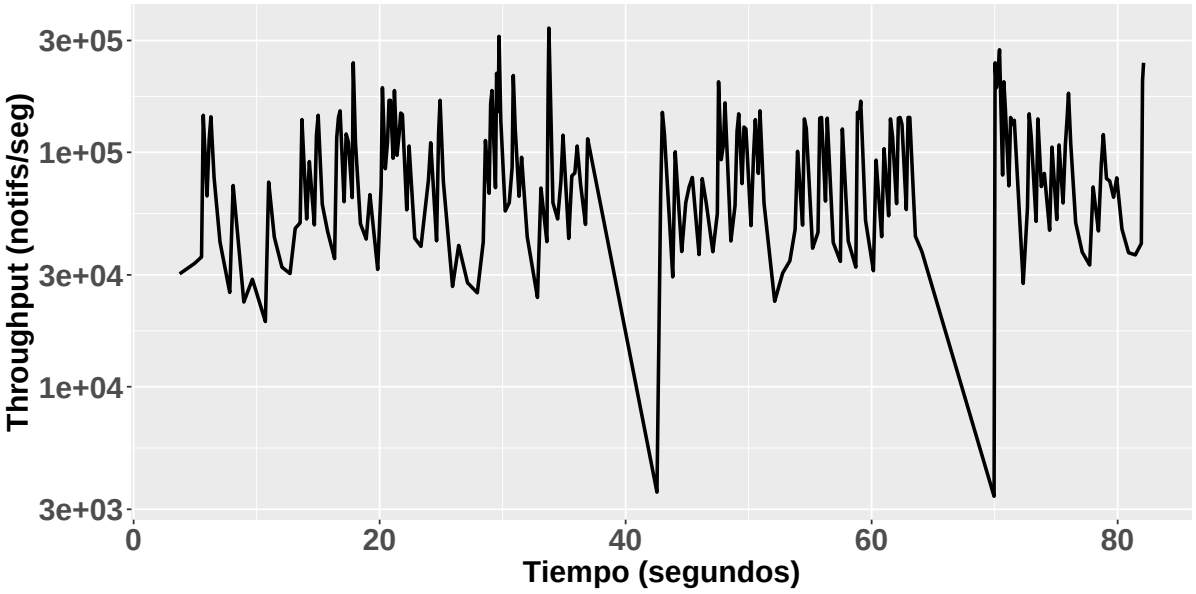

Figura 3.14: Throughput de prueba con carga completa e input rate de 500.000 mensajes/s.

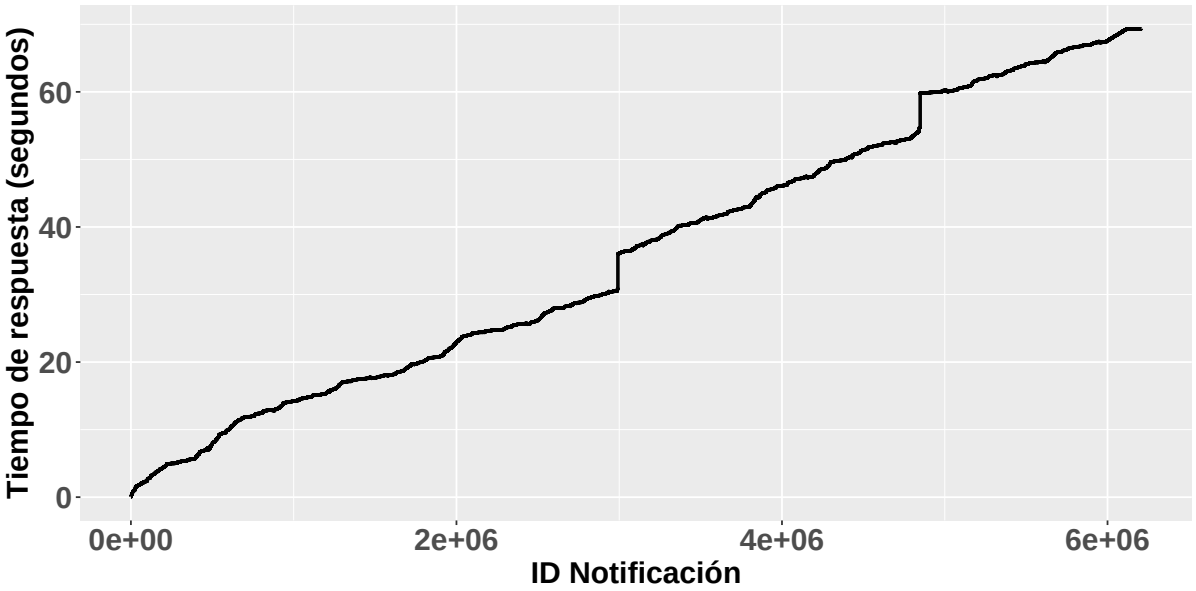

Figura 3.15: Tiempo de respuesta de prueba con carga completa e input rate de 500.000 mensajes/s.

valores que en las pruebas anteriores.

Conclusiones de la prueba:

- El throughput presenta serias caídas (hasta los 3.000 mensajes por segundo), y no llega a los 300.000 mensajes por segundo.
- El tiempo de respuesta denota la saturación del sistema, llegando hasta los 80 segundos de espera en la llegada de las notificaciones.

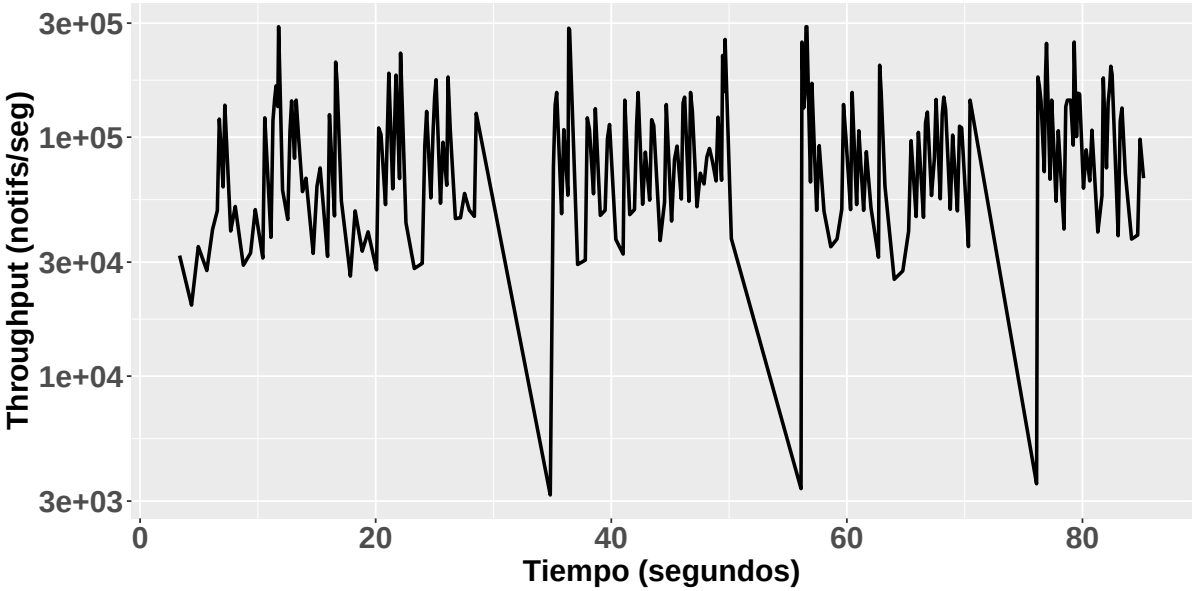

Figura 3.16: Throughput de prueba con carga completa e input rate de 1.000.000 mensajes/s.

- 6.433.794 mensajes por segundo (máxima velocidad posible)

Esta prueba presenta unos resultados similares a los resultados de las anteriores pruebas (lo cual es esperable), con un throughput con caídas (ver Figura 3.18), que coinciden con incrementos en el tiempo de respuesta (ver Figura 3.19), el cual presenta unos valores en incremento, igual que en previas pruebas, llegando a los 85 segundos de tiempo de respuesta.

Conclusiones de la prueba:

- Viendo esta prueba y las anteriores, el throughput no supera los 300.000, y en este caso, solo supera los 200.000 en ciertos picos, precedidos de valles (notificaciones encoladas que se procesan a gran velocidad).
- El tiempo de respuesta presenta unos valores en incremento, lo que concuerda con los valores del throughput, y con la propia saturación del sistema.

Secuencia logarítmica de subscripciones

Como se puede observar en la Figura 3.20, el throughput del sistema decae de forma considerable pasadas las 20.000 subscripciones activas, ya que el tiempo invertido

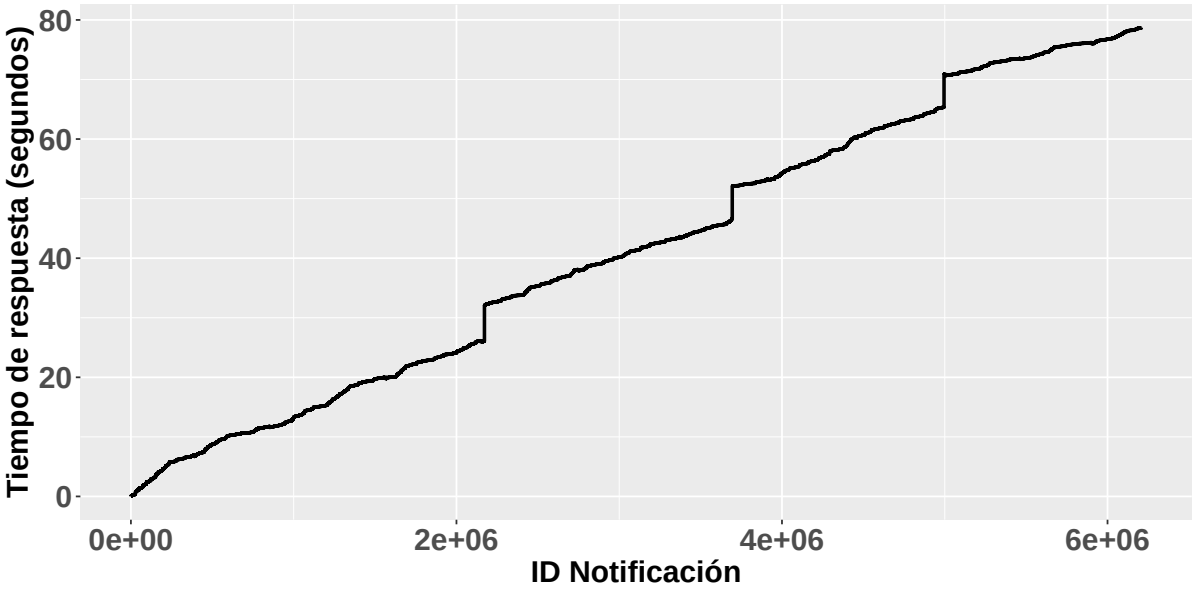

Figura 3.17: Tiempo de respuesta de prueba con carga completa e input rate de 1.000.000 mensajes/s.

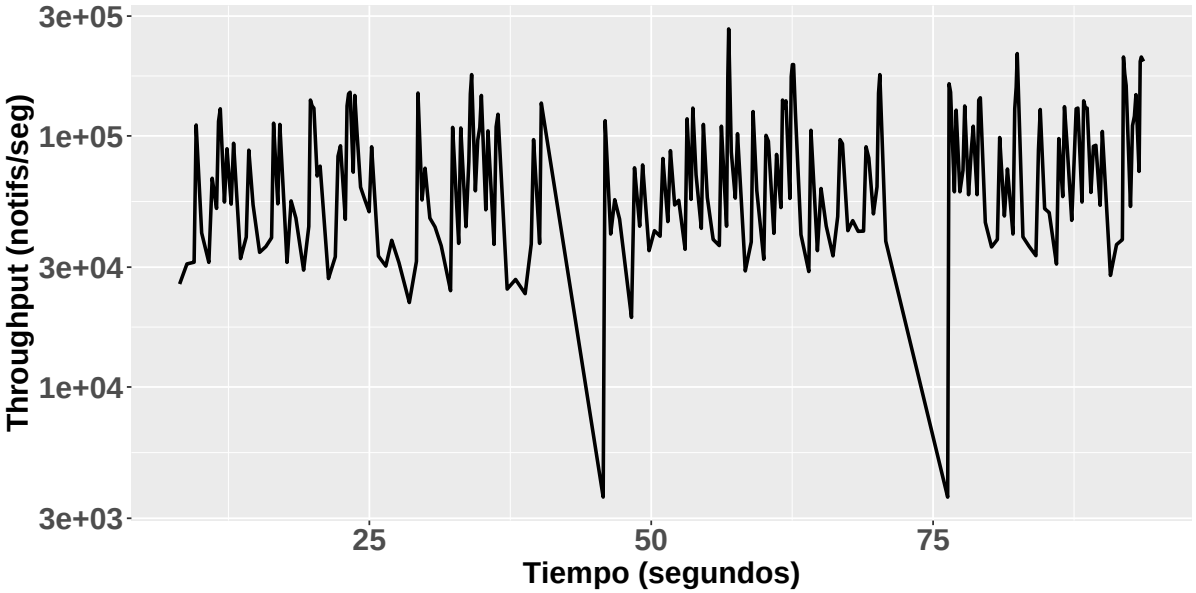

Figura 3.18: Throughput de prueba con carga completa a máximo input rate.

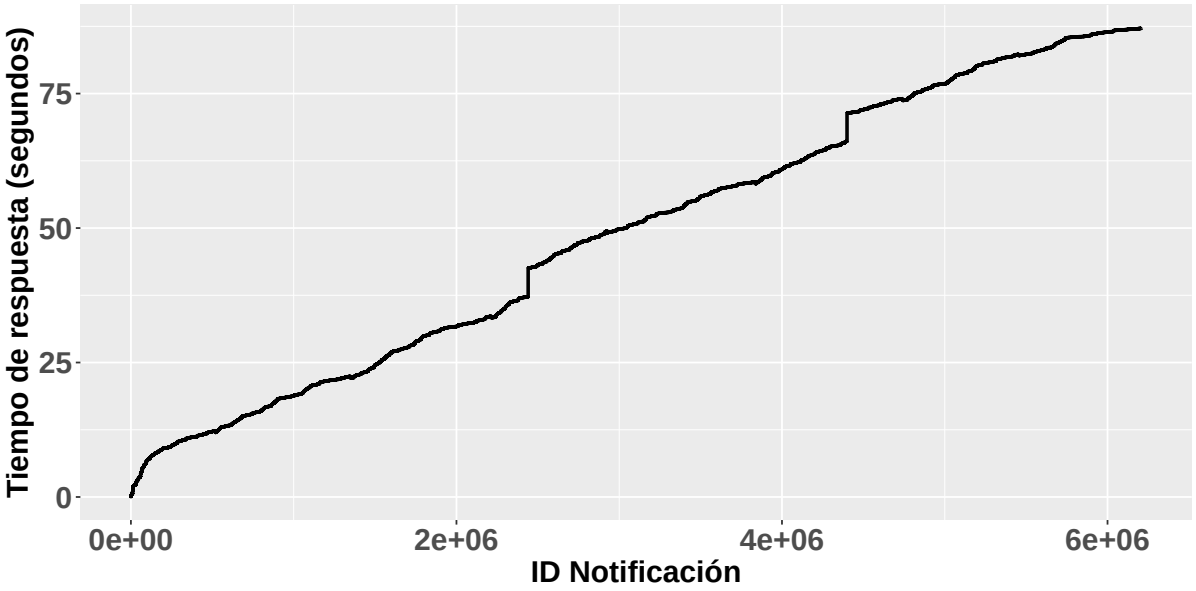

Figura 3.19: Tiempo de respuesta de prueba con carga completa a máximo input rate.

en comprobar todas las subscripciones incrementa y provoca un aumento en el tiempo de procesado de cada publicación, en concreto, en la confección de la lista de subscripciones interesadas en dicha publicación.

Esto indica que con la configuración actual (1-1-1), el sistema está saturando cuando tiene que comprobar, al menos, 20.000 subscripciones. En este punto, si estuviese activada la escalabilidad, el sistema debería de haber escalado para no llegar a saturar.

Conociendo este punto de saturación, lo siguiente es probar el sistema con esa carga de subscripciones, de forma que se vea con mayor precisión cómo actúa el sistema con un número de subscripciones cercano al del punto de saturación.

Carga de subscripciones creciente

Como se puede observar en la Figura 3.21, el throughput del sistema cae en gran medida, con una media de 18.000 mensajes por segundo, con picos cercanos a los 30.000.

Este primer pico se debe al aumento de subscripciones, que provocan que las publicaciones enviadas de forma paralela generen un mayor número de notificaciones, lo cual aumenta el número de notificaciones generadas por segundo.

De igual forma, estos picos pueden estar causados por la naturaleza de la distribución de subscripciones y publicaciones, ya que ciertas secciones de la carga de publicaciones pueden generar mayor número de notificaciones, al machear con más subscripciones activas.

A causa de esto, se requiere de realizar nuevas pruebas con otro tipo de carga de trabajo, que pruebe la saturación del sistema con incrementos puntuales, como se puede esperar en un contexto real.

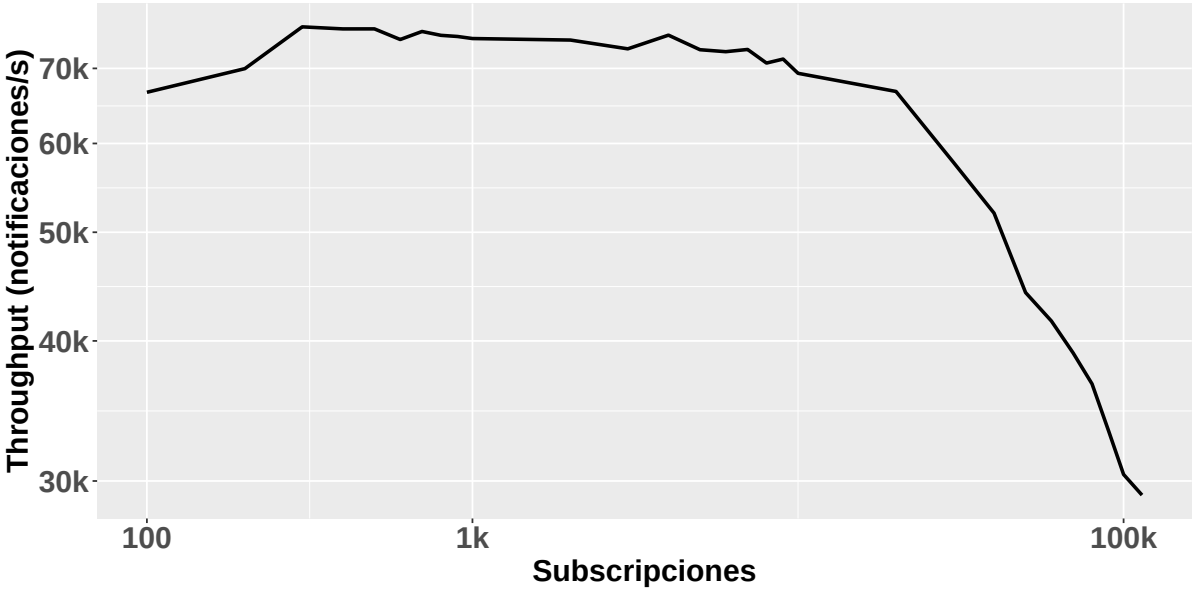

Figura 3.20: Throughput en base a la subscripciones activas con un input rate de 100.000 mensajes por segundo en ejes logarítmicos.

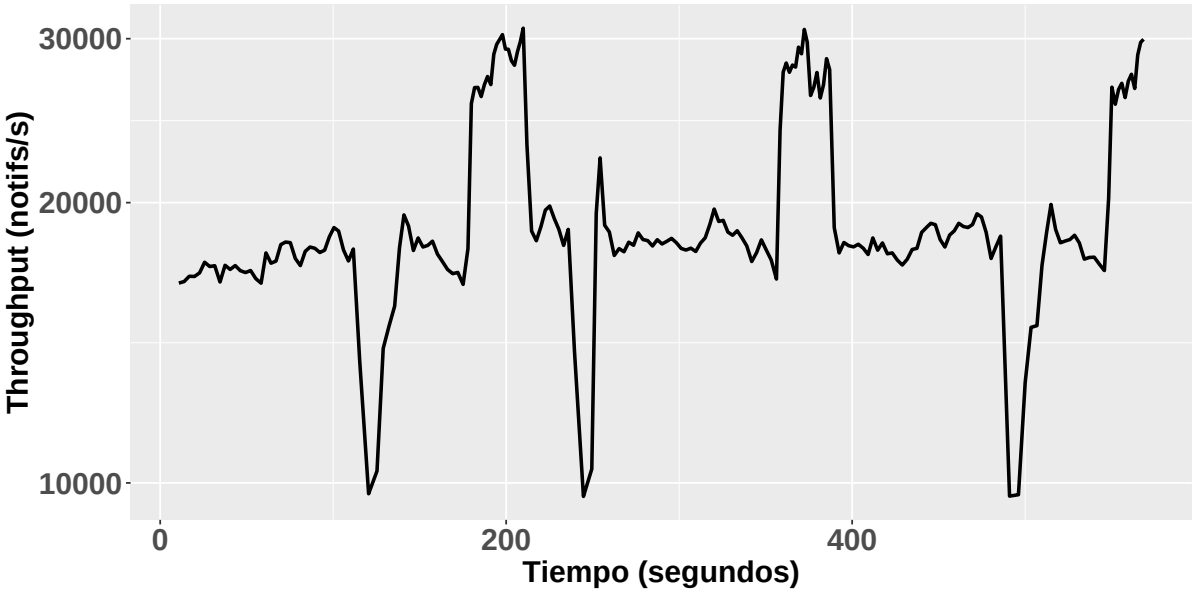

Figura 3.21: Throughput del sistema con la carga de subscripciones y todas las publicaciones

Carga de subscripciones estática

- 50.000 subscripciones

En la Figura 3.22 se puede observar la caída de rendimiento del sistema, pues ha llegado al punto de saturación parcial al caer este muy por debajo del throughput esperado de 200.000 mensajes por segundo. Las fluctuaciones indican que el sistema, en ciertos momentos, encola gran cantidad de eventos, al no poder procesarlos a tiempo, pero recupera ese tiempo perdido en los siguientes segundos. A pesar de esto, el sistema no llega a saturar de la forma esperada, es decir, se recupera más rápido de lo esperado, lo cual es bueno para el sistema, pero no nos proporciona la información buscada.

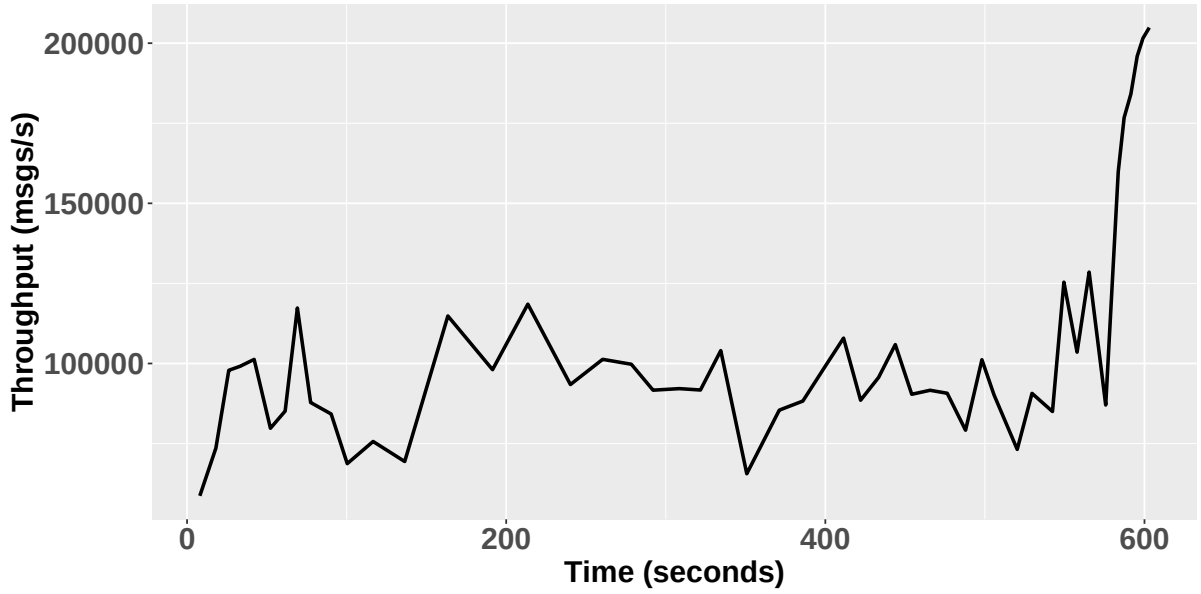

Figura 3.22: Throughput resultante de una prueba con 50.000 subscripciones y un input rate de 200.000 publicaciones por segundo.

- 100.000 subscripciones

Si se lleva esta prueba al límite, es decir, el caso máximo en el cual dicha prueba puede acabar sin quedarse el ordenador sin memoria para llevarla a cabo, se puede observar la inestabilidad del rendimiento del sistema, al haber este saturado de forma parcial. Esto se puede observar en la Figura 3.23.

Los resultados obtenidos mediante estas pruebas nos muestran que el sistema satura parcialmente de forma efectiva con una carga de 100.000 subscripciones y un input rate mayor o igual a 150.000 mensajes por segundo. Esta respuesta nos ayuda a encontrar, con mayor precisión, pero para ello se han de poner en práctica más pruebas con diferentes cargas.

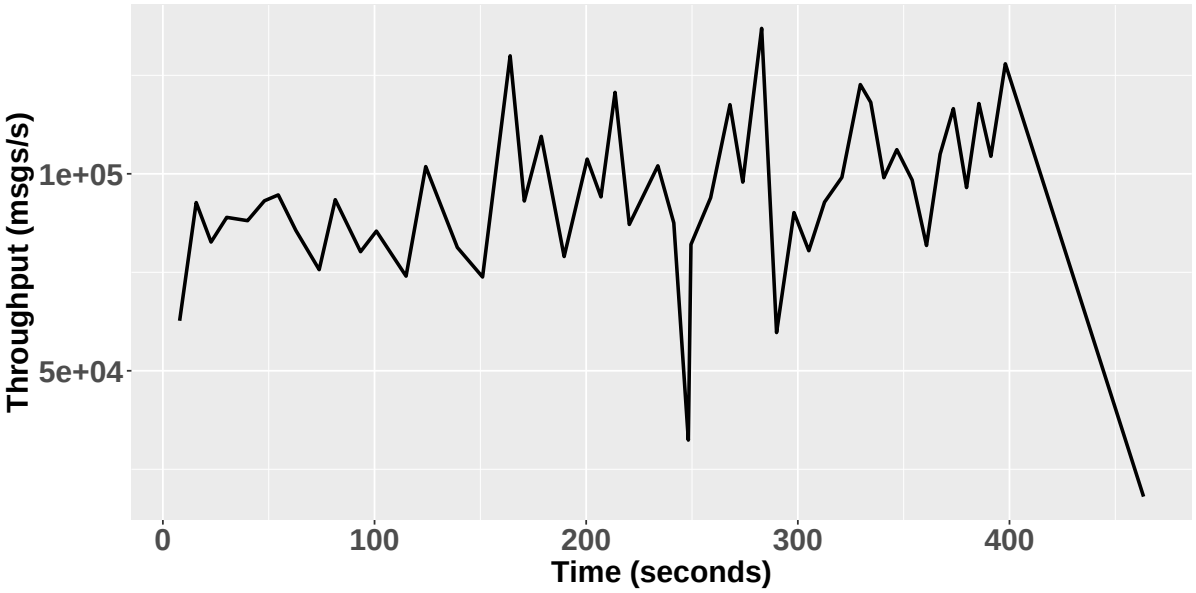

Figura 3.23: Throughput resultante de una prueba con 100000 subscripciones y un input rate de 200.000 publicaciones por segundo.

Carga de subscripciones con crecimiento puntual

Crecimiento de 20.000 a 100.000 subscripciones

Siguiendo el modelo de pruebas previamente empleado, mediante aplicar carga representada por la Figura 3.5, y especificando un input rate fijo de 125.000 mensajes por segundo, se ha obtenido el throughput que se muestra en la Figura 3.24.

El throughput de la Figura 3.24 demuestra que el sistema ha saturado parcialmente, y ha llegado a recuperarse en la segunda mitad de la prueba, llegando a un throughput de 125.000, al haber encolado gran cantidad de mensajes. Junto con esto, se aprecia un throughput muy inestable, con muchos picos bajos y altos. Los valores proporcionados para esta prueba son los más altos posibles, sin que la ordenador se quede sin memoria para poder llevar a cabo esta, por lo que sabemos que el sistema satura de forma efectiva a partir de estos valores.

Crecimiento de 25.000 a 100.000 subscripciones

A raíz de estos resultados, se ha desarrollado la carga representada por la Figura 3.6 que resulta en el throughput representado por la Figura 3.25.

Dados estos resultados, el comportamiento del sistema con estas pruebas no es el óptimo para aplicar estos valores a los modelos predictivos. A causa de esto, se ha desarrollado una prueba adicional, similar a las anteriores, pero que implementa un incremento más repentino (llegar al máximo en 2-3 segundos) con un menor input ratio, de forma que el sistema sature parcialmente en el punto del incremento, pero sea capaz de recuperarse a tiempo.

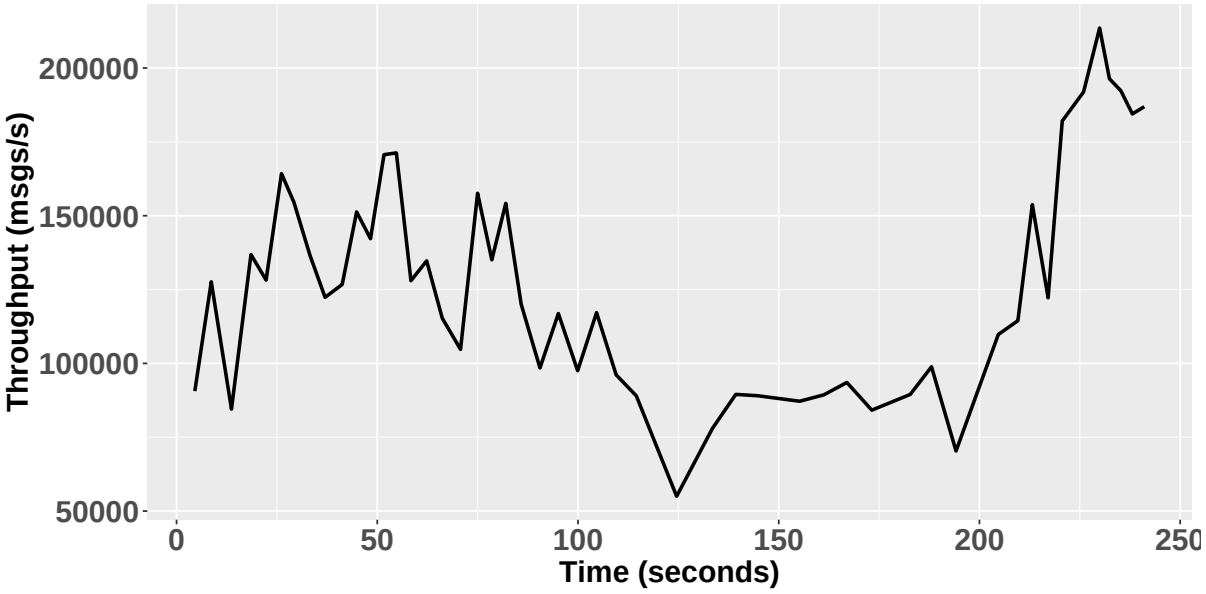

Figura 3.24: Throughput resultante de la carga de la Figura 3.5 e input rate de 125.000 publicaciones por segundo.

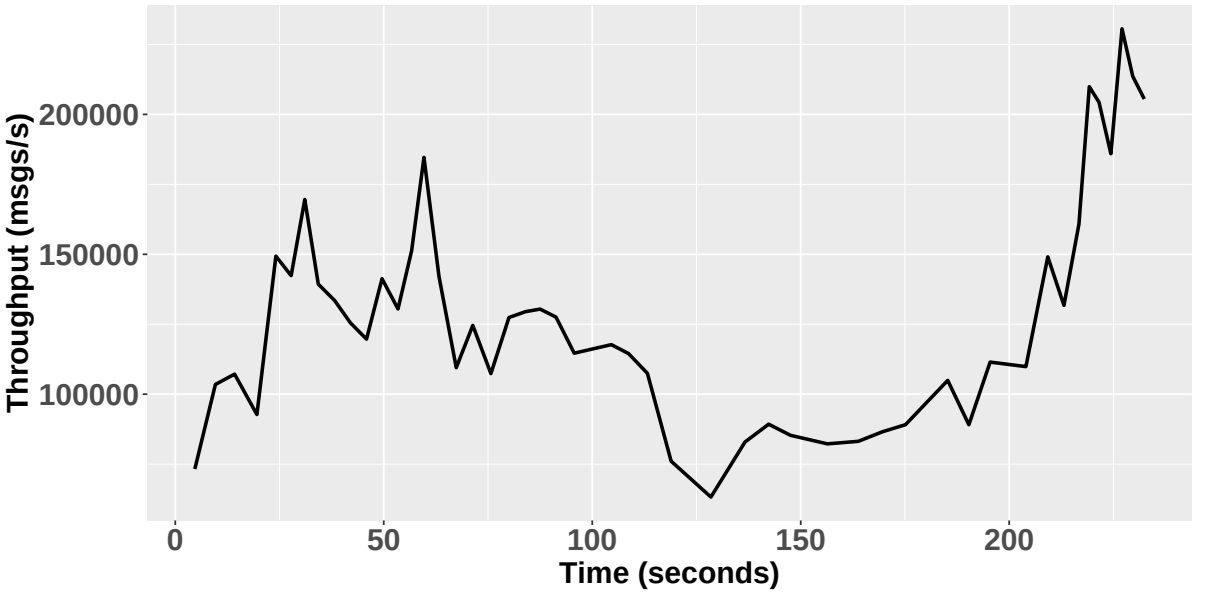

Figura 3.25: Throughput resultante de la carga de la Figura 3.6 e input rate de 125.000 publicaciones por segundo.

Crecimiento rápido de 25.000 a 113.904 subscripciones

Los resultados de esta prueba, que sigue los mismos parámetros que la anterior pero con un input ratio de 100.000 mensajes por segundo, se pueden observar en la Figura 3.26, que muestra una saturación muy leve del sistema cuando se alcanza el incremento de la carga, pero en sistema se recupera en muy poco tiempo.

Este resultado, a pesar de haber saturado de forma mínima el sistema, es el buscado, pues el throughput del sistema es estable y el sistema satura de la forma esperada, a pesar de haberse recuperado rápidamente.

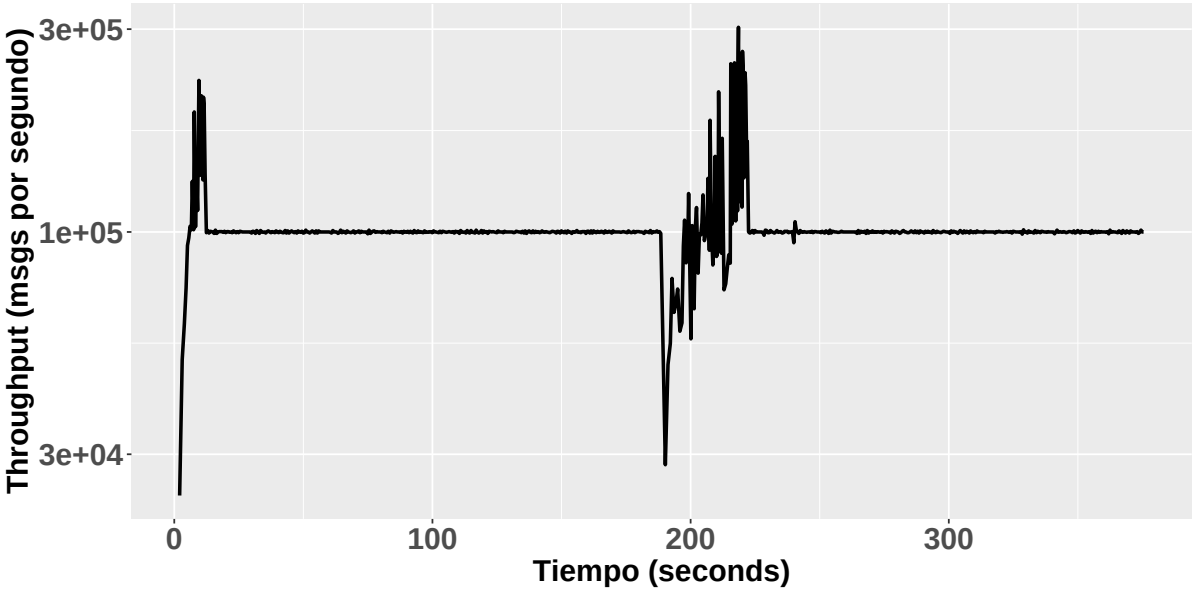

Figura 3.26: Throughput resultante de la carga de la Figura 3.7 e input rate de 100.000 publicaciones por segundo.

Habiendo obtenido estos resultados, que simulan diferentes situaciones reales, podemos usar estos para aplicar los modelos predictivos, una vez desarrollados e implementados estos. Los modelos predictivos usarán estos resultados para llevar a cabo dichas predicciones, para que el sistema pueda anticiparse a cualquier situación que sea posible predecir.

Capítulo 4

Generación de modelos predictivos para el auto-escalado de E-SilboPS

En este capítulo, se aborda la generación de los modelos predictivos, mediante la implementación de algoritmos de Machine-Learning y Deep-Learning, que usarán los resultados de las pruebas de rendimiento para predecir el comportamiento del sistema.

4.1. Modelos predictivos

Además de estos modelos, también se ha implementado la capacidad de calcular la primera derivada de los resultados de los modelos, pudiendo obtener resultados más detallados.

4.1.1. Métodos de Series Temporales

Los primeros modelos que se han implementado han sido los de Series Temporales, que hacen uso de "Rolling Forecasting Origin"[19] para ajustar dichos modelos y poder realizar predicciones.

Para comprobar los resultados de estos modelos, se comparan los valores de la predicción con todos los anteriores, no solo con el inmediatamente anterior, lo que lleva a un mejor ajuste de estos modelos al aumentar la precisión de dichas predicciones.

Rolling Forecasting Origin

Este método permite dividir una serie de datos temporales en dos series de datos, una de entrenamiento, y otra de pruebas, de forma que los modelos se puedan ajustar con la primera, y se prueben con la segunda.

Esta división se lleva a cabo mediante especificar el tamaño inicial de cada serie, es decir, el número de valores de la serie temporal que se usarán de forma inicial; y variando el horizonte, que es el número valores consecutivos en la serie de prueba.

De esta forma, y mediante el tamaño inicial, se establecen el número de repeticiones de este método ("resampling"), aumentando/moviendo las series temporales de entrenamiento y pruebas, tal como se ve en la Figura 4.1.

Adicionalmente, se puede establecer que la serie de entrenamiento añada los nuevos valores a los anteriores, de forma que esta crezca con cada repetición.

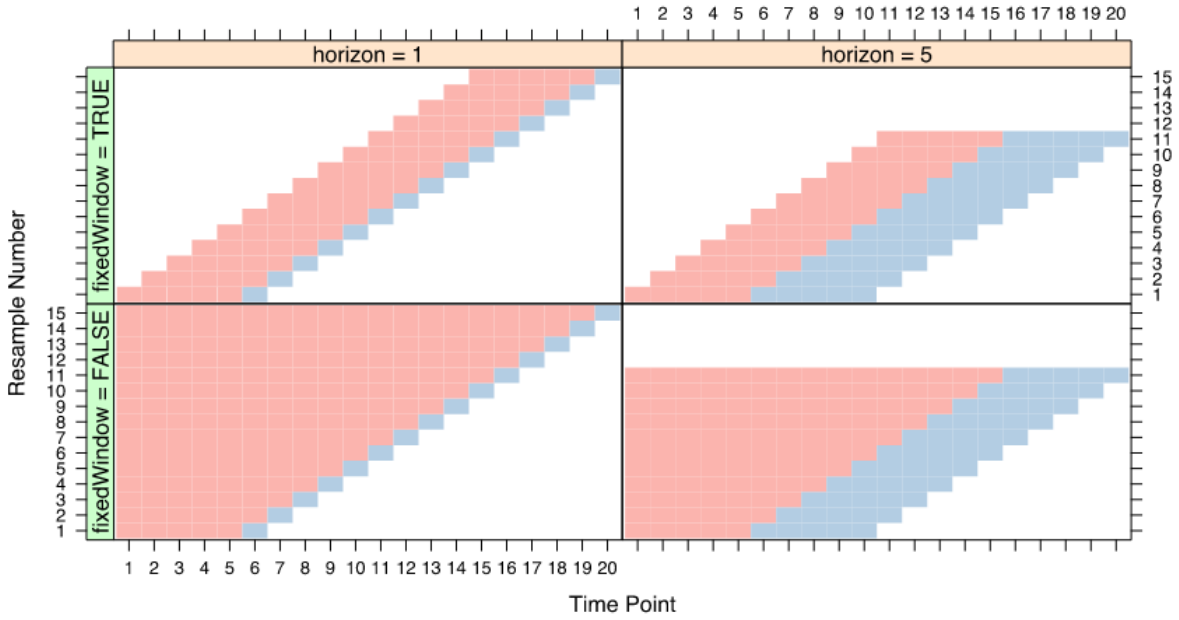

Figura 4.1: Aplicación de "Rolling Forecasting Origin" a una serie de 20 elementos con un tamaño inicial de cada serie de 5 elementos. Los elementos en rojo pertenecen a la serie de entrenamiento, los azules a la serie de pruebas. Imagen obtenida de [19]

Modelos ARIMA

La implementación del modelo ARIMA, acrónimo del inglés Autoregressive Integrated Moving Average, y generalización del modelo ARMA, acrónimo del inglés Autoregressive Moving Range; permite, mediante el uso de la variación y regresión de los datos estadísticos, encontrar patrones en dichos datos para poder realizar predicciones a futuro.

Este proceso se lleva a cabo mediante ajustar el modelo con una serie de datos de entrenamiento, tras lo que se procede a comprobar si el modelo se ajusta de forma correcta. Una vez ajustado y comprobado el modelo, este puede generar nuevas predicciones.

Modelos STL con ETS[1]

También se ha implementado el modelo STL, acrónimo del inglés "Seasonal and Trend decomposition using Loess", para descomponer las series temporales junto con el modelo ETS, acrónimo del inglés "Error, Trend and Seasonal".

Estos modelos obtienen predicciones de los datos ajustados estacionalmente mediante aplicar a estos predicciones no estacionales, y reajustando esa estacionalidad

[1] https://otexts.com/fpp2/weekly.html

usando los últimos datos de la serie de datos originales (ajustados estacionalmente).

4.1.2. Modelos de Machine-Learning y Deep-Learning

Modelos de Regresión Lineal

Los modelos de Regresión Lineal utilizan las dos variables presentes en los datos, en este caso, throughput y tiempo, para aproximar la relación de dependencia entre ambas variables.

Una vez conocida la relación entre ambas variables, el modelo puede predecir futuros valores en base a dicha relación.

Modelos Generalizados Aditivos

Los Modelos Generalizados Aditivos son una extensión de los Modelos Lineales Generalizados, y consideran que la relación entre las variables respuesta (la variable que queremos predecir, en este caso, el throughput) y las variables explicativas (la variable que se utiliza para realizar las predicciones, en este caso, el tiempo) puede tomar cualquier forma funcional, no únicamente lineal.

Mediante esta consideración, estos modelos toman los datos y realizan predicciones mediante la aplicación de la relación entre ambas variables y las funciones que pueden representar.

Modelos Generalizados Lineales

Los Modelos Generalizados Lineales generalizan la Regresión Lineal permitiendo que el modelo lineal se relacione con la variable de respuesta a través de una función de enlace y permitiendo que la magnitud de la varianza de cada medida sea una función de su valor predicho.

Esto permite al modelo realizar predicciones mediante aplicar los datos ya obtenidos y la función de enlace, que depende de la distribución de los propios datos.

Modelos de "Random Forest"

Los modelos de "Random Forest" aplican los datos a un algoritmo de Machine-Learning que combina múltiples árboles de decisión clasificando y aplicando regresión a dichos datos.

Este algoritmo se entrena aprendiendo a clasificar los datos obtenidos, de forma que el "random forest" crece conforme aprende a clasificar nuevos elementos, de forma que con nuevos datos no clasificados, sea capaz de predecir su clasificación, es decir, el algoritmo, una vez entrenado, será capaz de predecir el comportamiento del sistema al clasificar los nuevos datos recibidos.

Modelos de Redes Neuronales

De forma similar a los modelos de "Random Forest", los modelos de redes neuronales utilizan los datos obtenidos para entrenar una red neuronal que, una vez entrenada, podrá predecir el comportamiento del sistema mediante clasificar los nuevos valores.

De forma adicional, este programa permite aplicar la primera derivada previa aplicación de estos modelos predictivos, pudiendo obtener la pendiente de las predicciones realizadas.

4.2. Resultados esperados

En el momento de la escritura de este Trabajo de Fin de Grado, debido a problemas de tiempo, causados por retrasos en el desarrollo de pruebas y obtención de resultados previos, no se ha podido aplicar y probar estos modelos de predicción.

A pesar de esto, al conocer el funcionamiento de estos modelos, se pueden esperar ciertos resultados de la aplicación de los mismos a este trabajo.

Estos resultados esperados mostrarán una predicción del comportamiento del sistema en base a los valores anteriores, generando una gráfica en la que se mostrarán los datos anteriores junto con las predicciones de los modelos, generando un gráfico similar al de la Figura 4.2.

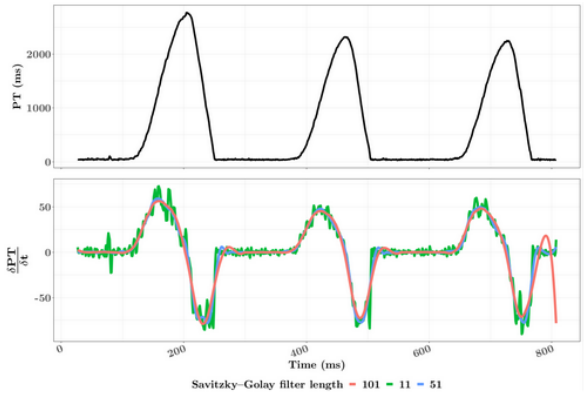

Figura 4.2: Ejemplo de resultado de los modelos predictivos.

Capítulo 5

Impacto del trabajo

En este capítulo se hace un análisis del impacto general del trabajo desarrollado en este TFG, así como el impacto del mismo conforme a los Objetivos de Desarrollo Sostenible.

5.1. Impacto general

Los sistemas distribuidos publicador/subscriptor se encuentran en una etapa de grandes avances y desarrollos, pues son una tecnología ampliamente usada al vivir en un mundo cada día más digitalizado.

De igual forma, la computación en la nube ha sufrido grandes avances, al proporcionar una plataforma en la que los usuarios pueden desplegar sus sistemas y pagar solo por los recursos usados, a diferencia de la tradicional forma de desplegar sistemas de cara al público mediante tener recursos de computación estáticos, los cuales puedes estar desaprovechados y conllevan mayores costes.

Mediante el uso de computación en la nube y aplicando un buen sistema de auto-escalado, estos sistemas publicador/subscriptor proporcionarán un servicio eficiente minimizando el coste de despliegue y mantenimiento.

En este proyecto, desde su inicio con la implementación del sistema SilboPS[3] y siguiendo con la posterior mejora de este, que ha resultado en el sistema E-SilboPS[1][2], se ha desarrollado un sistema publicador/subscriptor que implemente, de forma óptima y eficiente, la función de auto-escalado, de forma que el sistema requiera de mínima supervisión y cumpla con lo previamente mencionado.

Mediante el desarrollo de este proyecto se pretende contribuir al avance y mejora de los sistemas publicador/subscriptor por medio de auto-escalado eficiente, lo que lleva a proporcionar un mejor servicio a los usuarios de estos sistemas, de forma que el tiempo de respuesta en el que reciben las publicaciones sea mínimo, incluso en situaciones de mucho tráfico en el sistema, a la vez que se minimiza el uso de los recursos de computación, reduciendo los costes y el malgasto energético al no desperdiciar dichos recursos.

El trabajo llevado a cabo en este TFG, que complementa el ya realizado en los trabajos previamente mencionados, mediante el estudio del comportamiento del sistema frente a una situación que refleja una real, y los modelos de predicción, se pretende aplicar

mejoras al auto-escalado de dicho sistema, aumentando su eficiencia. Este estudio se ha llevado a cabo sobre una configuración básica 1-1-1, de forma que se identifique el punto de saturación del sistemas bajo esta configuración, resultado el cual, mediante la aplicación de modelos de predicción, permitirá la implementación de mejoras del auto-escalado.

Este trabajo ha permitido establecer las bases para el desarrollo de futuras mejoras del auto-escalado del sistema E-SilboPS, las cuales se podrán implementar en otros sistemas publicador/subscriptor del mismo tipo.

5.2. Objetivos de Desarrollo Sostenible

El impacto de este trabajo sobre los 17 Objetivos de Desarrollo Sostenible aprobados en 2015 por los Estados Miembros de las Naciones Unidas como parte de la Agenda 2030 para el Desarrollo Sostenible[20], que se pueden ver en la Figura 5.1, se alinea con el **Objetivo 9. Industria, innovación e infraestructura**, el **Objetivo 11. Ciudades y Comunidades Sostenibles** y con el **Objetivo 12. Producción y Consumo Responsable**.

Figura 5.1: Objetivos de Desarrollo Sostenible (*ODS*). Imagen obtenida de [20].

Los sistemas publicador/subscriptor, como el tratado en este trabajo, están en constante desarrollo e innovación, y son utilizados en multitud de ámbitos y casos de uso en la vida cotidiana, por lo que sus constantes mejoras son vitales para el funcionamiento óptimo de estos sistemas. De igual forma, estas mejoras conllevan un menos consumo eléctrico, al requerir de menos operaciones por cada evento, o de desplegar un menor número de sistemas publicador/subscriptor si cada un puede con más carga.

A causa de esto, esta tecnología y este trabajo se alinean principalmente con el **Objetivo 9. Industria, innovación e infraestructura**, y de forma más precisa, con las metas **9.1**, al crear una infraestructura sostenible; **9.2** debido al constante desarrollo e innovación de la tecnología; y **9.4**, ya que el objetivo es modernizar los sistemas publicador/subscriptor y hacerlos maś eficientes.

De igual forma, se alinean, del **Objetivo 11. Ciudades y comunidades sostenibles**, con las metas **11.3** Urbanización inclusiva y sostenible, al desarrollar sistemas informáticos sostenibles medio-ambientalmente mediante la reducción del consumo energético de estos sistemas, ya sea por la mayor eficiencia de las operaciones del sistema o por la eficiencia del sistema completo, requiriendo de menos sistemas de este tipo al poder aumentar la carga de cada uno, requiriendo de menor número de sistemas desplegados.

Por último, este trabajo, al reducir el consumo energético de estos sistemas mediante aumentar la eficiencia, también están relacionadas con las metas del **Objetivo 12. Producción y consumo responsables**, principalmente, con la **12.A** Ciencia y Tecnología para Sostenibilidad, al desarrollar un sistema que puede ser usado para impulsar la transformación digital y llevar esta tecnología a lugares poco digitalizados.

El impacto medioambiental del trabajo desarrollado en este TFG es mínimo, pero una vez cumplido el objetivo último de este proyecto (maximizar la eficiencia del auto-escalado de los sistemas publicador/subscriptor basados en contenido mediante E-SilboPS), si estas mejoras se implantasen de forma generalizada en la sociedad, junto con la constante digitalización de la vida cotidiana, se podría reducir la contaminación producida por estos sistemas en gran medida, al ser estos más eficientes.

De forma adicional, este proyecto tiene un potencial impacto en todas aquellas áreas que se beneficiarían de una mayor sostenibilidad mediante la transformación digital resultante de implementar y usar este tipo de sistemas, principalmente, en el Internet of Things y en Smart Cities.

Capítulo 6

Resultados y conclusiones

En este capítulo final, se presentarán los resultados de este trabajo, las conclusiones personales del autor y el trabajo futuro del proyecto.

6.1. Resultados

Los resultados obtenidos de este trabajo muestran que, mediante la aplicación de la carga de trabajo real obtenida y la aplicación de los modelos predictivos, se pueden obtener mejoras en el proceso de auto-escalado.

De igual forma, se ha implementado y probado de forma satisfactoria el generador de cargas de trabajo real, lo que permite que si en un futuro se dispone de alguna otra carga real que pudiese ser de utilidad. Esto se consigue mediante el cambio en la configuración a la sintaxis correspondiente de la nueva carga de trabajo.

Como se ha mencionado en la Sección 4.1, al no haber podido aplicar los modelos de predicción, no se han podido obtener los resultados esperados, pero los conocimientos adquiridos durante la realización de este trabajo, permitirán aplicar estos modelos predictivos para aplicar las mejoras necesarias al sistema.

En general, los resultados han sido positivos, a pesar de no haber tenido tiempo suficiente para poder llevar a cabo todas las tareas que se habían planteado en un principio, debido principalmente a retrasos generados por problemas durante el desarrollo y el análisis de los datos del sistema.

6.2. Conclusiones personales

El desarrollo de este proyecto ha sido muy gratificante, tanto personalmente como académicamente, y me ha permitido aprender y desarrollarme como estudiante, desde leer y entender documentos científicos hasta desarrollar código de forma eficiente y profesional, poniendo en práctica tanto conocimientos obtenidos durante el grado como aquellos obtenidos durante el desarrollo de este TFG.

Personalmente, me ha gustado mucho poder conocer y participar en el desarrollo de este proyecto tan ambicioso y llamativo, así como conocer el estado actual de las

tecnologías relacionadas, tanto directa como indirectamente. El futuro de este proyecto y los posibles resultados que se pueden obtener me resultan muy interesantes y emocionantes.

6.3. Trabajo futuro

El trabajo futuro de este proyecto, principalmente, abarca la aplicación de los modelos de predicción, en concreto, el de una carga incremental (se incrementa la carga de permanente), una carga puntual (incremento puntual de la carga), y el de una carga no predecible (la carga se genera de forma aleatoria, en base a ciertos parámetros). Tras aplicar estos modelos, se analizarán los resultados obtenidos y se compararán con los resultados de los actuales modelos estadísticos de predicción para ver si se producen mejores predicciones con estos modelos predictivos.

Siguiendo la aplicación de los modelos predictivos, las mejoras que se identifiquen a partir de los resultados de estos serán otro de los trabajos a realizar en un futuro.

De forma adicional, sería recomendable revisar y actualizar los paquetes y dependencias del proyecto, de forma que haya estabilidad y uniformidad en las versiones usadas, y se disponga de las últimas actualizaciones y mejoras de las herramientas usadas, además de prevenir posibles vulnerabilidades, como la recientemente vulnerabilidad de Log4J[21].

Finalmente, otra de las tareas futuras es la de unificar el proyecto E-SilboPS en uno solo, al estar actualmente compuesto por dos proyectos, integrando ambos en uno solo y resolviendo cualquier problema de compatibilidad o conflicto, de forma que este sea más manejable y pueda documentarse de forma extensa en el futuro.

Bibliografía

[1] Víctor Rampérez Martín. «Improving elasticity in publish/subscribe systems for large-scale IoT based services». Tesis de mtría. ETSI_Informatica - Universidad Politecnica de Madrid, 2017. URL: http://oa.upm.es/51464/.

[2] Víctor Rampérez Martín. «A Technology-Agnostic Approach to Auto-Scale Services in Heterogeneous Clouds». 2021. URL: http://oa.upm.es/67259/.

[3] Sergio Vavassori. «A Novel Approach to Context-Aware Content-Based Middleware». Tesis doct. Universidad Politécnica de Madrid, 2016, pág. 147. DOI: 10.20868/UPM.thesis.39615. URL: http://oa.upm.es/39615/.

[4] Antonio Carzaniga, David S. Rosenblum y Alexander L. Wolf. «Design and Evaluation of a Wide-Area Event Notification Service». En: *ACM Trans. Comput. Syst.* 19.3 (ago. de 2001), págs. 332-383. ISSN: 0734-2071. DOI: 10.1145/380749.380767. URL: https://doi.org/10.1145/380749.380767.

[5] Patrick Th. Eugster y col. «The Many Faces of Publish/Subscribe». En: *ACM Comput. Surv.* 35.2 (jun. de 2003), págs. 114-131. ISSN: 0360-0300. DOI: 10.1145/857076.857078. URL: https://doi.org/10.1145/857076.857078.

[6] Raphaël Barazzutti y col. «Elastic Scaling of a High-Throughput Content-Based Publish/Subscribe Engine». En: *2014 IEEE 34th International Conference on Distributed Computing Systems*. 2014, págs. 567-576. DOI: 10.1109/ICDCS.2014.64.

[7] Universidad Politécnica de Madrid. *Information and Communications Technology Research Group: CETTICO*. URL: https://www.upm.es/recursosidi/en/map/en_information-and-communications-technology-research-group-cettico/.

[8] E. Fidler y col. «The PADRES Distributed Publish/Subscribe System.» En: *In 8th International Conference on Feature Interactions in Telecommunications and Software Systems*. Ene. de 2005, págs. 12-30.

[9] Lukasz Opyrchal y col. «Exploiting IP Multicast in Content-Based Publish-Subscribe Systems». En: *Middleware 2000*. Ed. por Joseph Sventek y Geoffrey Coulson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, págs. 185-207. ISBN: 978-3-540-45559-2.

[10] J. M. Bacon y P. R. Pietzuch. «Hermes: A Distributed Event-Based Middleware Architecture». En: *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*. Los Alamitos, CA, USA: IEEE Computer Society, jul. de 2002, págs. 611-618. DOI: 10.1109/ICDCSW.2002.1030837. URL: https://doi.ieeecomputersociety.org/10.1109/ICDCSW.2002.1030837.

[11] César Cañas y col. «Publish/Subscribe Network Designs for Multiplayer Games». En: *Proceedings of the 15th International Middleware Conference*. Middleware '14. Bordeaux, France: Association for Computing Machinery, 2014,

págs. 241-252. ISBN: 9781450327855. DOI: 10.1145/2663165.2663337. URL: https://doi.org/10.1145/2663165.2663337.

[12] Nikolas Herbst, Samuel Kounev y Ralf Reussner. «Elasticity in cloud computing: What it is, and what it is not». En: *International Conference on Autonomic Computing* (ene. de 2013), págs. 23-27.

[13] Raphaël Barazzutti y col. «StreamHub: A Massively Parallel Architecture for High-Performance Content-Based Publish/Subscribe». En: *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*. DEBS '13. Arlington, Texas, USA: Association for Computing Machinery, 2013, págs. 63-74. ISBN: 9781450317580. DOI: 10.1145/2488222.2488260. URL: https://doi.org/10.1145/2488222.2488260.

[14] Albert Yu, Pankaj Agarwal y Jun Yang. «Generating Wide-Area Content-Based publish/subscribe workloads». En: (nov. de 2014).

[15] Tania Lorido-Botran, Jose Miguel-Alonso y Jose A. Lozano. «A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments». En: *Journal of Grid Computing* 12 (4 nov. de 2014), págs. 559-592. ISSN: 15729184. DOI: 10.1007/s10723-014-9314-7.

[16] Ali Yadavar Nikravesh, Samuel A. Ajila y Chung Horng Lung. «An autonomic prediction suite for cloud resource provisioning». En: *Journal of Cloud Computing* 6 (1 dic. de 2017). ISSN: 2192113X. DOI: 10.1186/s13677-017-0073-4.

[17] K. Mani Chandy y Leslie Lamport. «Distributed Snapshots: Determining Global States of Distributed Systems». En: *ACM Trans. Comput. Syst.* 3.1 (feb. de 1985), págs. 63-75. ISSN: 0734-2071. DOI: 10.1145/214451.214456. URL: https://doi.org/10.1145/214451.214456.

[18] César Cañas y col. «Publish/Subscribe Network Designs for Multiplayer Games». En: *Proceedings of the 15th International Middleware Conference*. Middleware '14. Bordeaux, France: Association for Computing Machinery, 2014, págs. 241-252. ISBN: 9781450327855. DOI: 10.1145/2663165.2663337. URL: https://doi.org/10.1145/2663165.2663337.

[19] Max Kuhn. *Sección de datos para Series Temporales - Documentación del paquete "caret"*. URL: https://topepo.github.io/caret/data-splitting.html#data-splitting-for-time-series.

[20] Organización de las Naciones Unidas. *Objetivos de Desarrollo Sostenible*. URL: https://www.agenda2030.gob.es/objetivos/home.htm.

[21] Cybersecurity e Infrastructure Security Agency. *Apache Log4j Vulnerability Guidance*. URL: https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance.

Apéndice A

Anexo

A continuación se muestra el código, las figuras y las métricas de interés procedentes del desarrollo de este proyecto[1].

A. Datos en crudo

Primeras 100 líneas de la traza resultante del generador

```
[
        {
                "eventType" : "S",
                "id" : "rmi://10.0.1.1:1099/Broker1-M23",
                "contextFunction" : {},
                "constraints" : {
                        "class:str" : [{"=" : "broadcast"}]
                }
        },
        {
                "eventType" : "S",
                "id" : "rmi://10.0.1.1:1099/Broker1-M24",
                "contextFunction" : {},
                "constraints" : {
                        "class:str" : [{"=" : "id7471312"}]
                }
        },
        {
                "eventType" : "P",
                "class:str" : "broadcast"
        },
        {
                "eventType" : "P",
                "class:str" : "broadcast"
        },
        {
                "eventType" : "P",
                "class:str" : "broadcast"
        },
        {
                "eventType" : "P",
                "class:str" : "broadcast"
        },
        {
                "eventType" : "P",
```

[1] Se puede acceder a las trazas de las pruebas, código de los programas desarrollados, y cualquier recurso utilizado en este TFG bajo petición al autor.

```
                "class:str" : "broadcast"
        },
        {
                "eventType" : "S",
                "id" : "rmi://10.0.1.1:1099/Broker1-M36",
                "contextFunction" : {},
                "constraints" : {
                        "class:str" : [{"=" : "__announce__"}]
                }
        },
        {
                "eventType" : "P",
                "class:str" : "__announce__"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "ProxyID_-3884500000",
                "x:str" : "0.0",
                "y:str" : "0.0"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "__announce__"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
                "eventType" : "P",
                "class:str" : "id11296229"
        },
        {
```


Primeras 100 líneas de la traza resultante que conforma la Figura 3.26

Anexo

```
"timestamp","throughput"
30.414,100401
30.912,100401
31.419,98619
31.917,100401
32.414,100603
32.913,100200
33.412,100200
33.919,98619
34.412,101419
34.917,99009
35.473,89928
35.913,113636
36.413,100000
36.917,99206
37.417,100000
37.918,99800
38.417,100200
38.914,100603
39.416,99601
39.917,99800
40.416,100200
40.913,100603
41.417,99206
41.916,100200
42.416,100000
42.916,100000
43.416,100000
43.916,100000
44.415,100200
44.916,99800
45.412,100806
45.916,99206
46.416,100000
46.912,100806
47.436,95419
47.912,105042
48.416,99206
48.915,100200
49.416,99800
49.916,100000
50.416,100000
50.914,100401
51.416,99601
51.938,95785
52.415,104821
52.918,99403
53.416,100401
53.916,100000
54.417,99800
54.916,100200
55.415,100200
55.913,100401
56.412,100200
56.915,99403
57.415,100000
57.913,100401
58.417,99206
58.912,101010
59.414,99601
59.917,99403
60.416,100200
60.916,100000
61.415,100200
61.916,99800
62.415,100200
62.916,99800
63.416,100000
63.913,100603
```

```
64.417,99206
64.915,100401
65.416,99800
65.915,100200
66.416,99800
66.916,100000
67.417,99800
67.916,100200
68.416,100000
68.916,100000
69.416,100000
69.915,100200
70.416,99800
70.916,100000
71.416,100000
71.912,100806
72.416,99206
72.916,100000
73.412,100806
73.944,93984
74.417,105708
74.916,100200
75.414,100401
75.916,99601
76.416,100000
76.916,100000
77.416,100000
77.915,100200
78.416,99800
78.915,100200
79.412,100603
```

B. Datos adicionales

Scripts en R

Listing A.1: Programa en R que genera una carga de subscripciones incremental, como en la Figura 3.3

```r
library(ggplot2)
library(dplyr)
library(tikzDevice)

# Function to export plot to LaTeX
print_LaTeX_plot <- function(path, plot, width, height) {
  tikz(filename = path, standAlone = TRUE, width = width, height = height)
  print(plot)
  dev.off()
  tools::texi2pdf(path, quiet = TRUE)
}

export_plots = TRUE
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

OUTPUT_CSV_DIR = "path/for/csv"
OUTPUT_PLOT_DIR = "/paht/for/plot/pdf"
low_workload  = 50000 # Stable system
high_workload = 75000 # System saturates


amount_noise = 0

total_duration = 60
windows_per_second = 10
total_windows = total_duration * windows_per_second

```

```
chunk_size = total_windows / 10

# First chunk (low steady)
first_chunk <- data.frame(subscriptions = rep(low_workload, chunk_size))

# Second chunk (increase)
step = (high_workload - low_workload) / (chunk_size * 2)
second_chunk <- data.frame(subscriptions = seq(low_workload, high_workload, step))

# Last chunk (high steady)
last_chunk <- data.frame(subscriptions = rep(high_workload, chunk_size * 7))

# Combine all chunks
workload <- rbind(first_chunk, second_chunk, last_chunk)

# Add noise
x <- workload$subscriptions
corrupt <- rbinom(length(x), 1, 1)     # choose an average of 10% to corrupt at random
corrupt <- as.logical(corrupt)
noise <- rnorm(sum(corrupt), 0, amount_noise) # generate the noise to add
x[corrupt] <- x[corrupt] + noise       # about 10% of x has been corrupted
workload$subscriptions <- x

plot <- ggplot(workload, aes(x = c(1:nrow(workload)))) +
  geom_line(aes(y = subscriptions), size=1) +
  xlab("Tiempo (segundos)") +
  ylab("Subscripciones") +
  theme(legend.position = "bottom",
        legend.text = element_text(size = 18, face="bold"),
        legend.title = element_text(size = 18, face="bold"),
        axis.text.x = element_text(size = 18, face="bold"),
        axis.text.y = element_text(size = 18, face="bold"),
        axis.title.x = element_text(size = 18, face="bold"),
        axis.title.y = element_text(size = 18, face="bold"))
plot

print_LaTeX_plot(OUTPUT_PLOT_DIR, plot = plot, width = 10, height = 5)

# Export
workload <- as.integer(workload$subscriptions)
write.table(workload, OUTPUT_CSV_DIR, col.names = FALSE, row.names = FALSE, sep = ";")
```

Listing A.2: Programa en R para obtener la carga representada en Figura 3.3

```
library(ggplot2)
library(tikzDevice)

# Function to export plot to LaTeX
print_LaTeX_plot <- function(path, plot, width, height) {
  tikz(filename = path, standAlone = TRUE, width = width, height = height)
  print(plot)
  dev.off()
  tools::texi2pdf(path, quiet = TRUE)
}
export_plots = TRUE
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))

##### Datos del plot #####

FILE_PATH = "../../logs/reports"
FILE_NAME = "logs-subs-10reps.csv"
XLABEL = "Subscripciones"
YLABEL = "Throughput (notificaciones/s)"
THEME <- theme(legend.position = "bottom",
               legend.text  = element_text(size = 18, face="bold"),
               legend.title = element_text(size = 18, face="bold"),
               axis.text.x  = element_text(size = 18, face="bold"),
               axis.text.y  = element_text(size = 18, face="bold"),
               axis.title.x = element_text(size = 18, face="bold"),
```

```
26                axis.title.y = element_text(size = 18, face="bold"))
27
28 ##### Codigo generador del plot #####
29
30 data <- read.csv(paste(FILE_PATH, FILE_NAME, sep="/"), header = TRUE, sep = ",")
31
32 plot <- ggplot(data, aes(x = subscriptions)) +
33   geom_line(aes(y = throughput), size=1) +
34   xlab(XLABEL) +
35   ylab(YLABEL) +
36   scale_y_log10(breaks = c(1, 30000, 40000, 50000, 60000, 70000, 80000),
37                      labels = c("1", "30k", "40k", "50k", "60k", "70k", "80k")) +
38   scale_x_log10(breaks = c(1, 10,100, 1000, 100000),
39                 labels = c("1", "10", "100", "1k", "100k")) +
40
41
42 print_LaTeX_plot("./throughput_logsubs.tex", plot = plot, width = 10, height = 5)
```

Listing A.3: Programa en R que genera una carga de subscripciones con un pico puntual, como en la Figura 3.5

```
1  library(ggplot2)
2  library(dplyr)
3  library(tikzDevice)
4
5  # Function to export plot to LaTeX
6  print_LaTeX_plot <- function(path, plot, width, height) {
7    tikz(filename = path, standAlone = TRUE, width = width, height = height)
8    print(plot)
9    dev.off()
10   tools::texi2pdf(path, quiet = TRUE)
11 }
12
13 export_plots = TRUE
14 setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
15
16
17 # SCRIPT PARAMETERS
18 OUTPUT_CSV_DIR = "path/for/csv"
19 OUTPUT_PLOT_DIR = "path/for/plot/pdf"
20 low_workload  = 25000  # Stable system
21 high_workload = 113904  # System saturates
22 fast_inc = TRUE
23 plain_spike = FALSE
24
25
26 amount_noise = 0
27
28 total_duration = 60
29 windows_per_second = 10
30 total_windows = total_duration * windows_per_second
31
32 chunk_size = total_windows / 10
33
34 # First chunk (low steady)
35 first_chunk <- data.frame(subscriptions = rep(low_workload, chunk_size*3))
36
37 # Second chunk (fast or slow increase)
38 step_up = (high_workload - low_workload) / ((chunk_size)*(fast_inc) ? 10 : 1)
39 second_chunk <- data.frame(subscriptions = seq(low_workload, high_workload, step_up))
40
41 if (plain_spike) {
42   # Extra chunk (top steady)
43   extra_chunk <- data.frame(subscriptions = rep(high_workload, chunk_size*0.5))
44 }
45
46 # Third chunk (decrease)
47 third_chunk <- data.frame(subscriptions = seq(high_workload, low_workload, -step_up))
```

```
48
49 # Last chunk (low steady)
50 last_chunk <- data.frame(subscriptions = rep(low_workload, chunk_size*3))
51
52 # Combine all chunks
53 if (plain_spike) {
54   workload <- rbind(first_chunk, second_chunk, third_chunk, last_chunk)
55 }else {
56   workload <- rbind(first_chunk, second_chunk, extra_chunk, third_chunk, last_chunk)
57 }
58
59 # Add noise
60 x <- workload$subscriptions
61 corrupt <- rbinom(length(x), 1, 1)    # choose an average of 10% to corrupt at random
62 corrupt <- as.logical(corrupt)
63 noise <- rnorm(sum(corrupt), 0, amount_noise) # generate the noise to add
64 x[corrupt] <- x[corrupt] + noise      # about 10% of x has been corrupted
65 workload$subscriptions <- x
66
67 plot <- ggplot(workload, aes(x = c(1:nrow(workload)))) +
68   geom_line(aes(y = subscriptions), size=1) +
69   xlab("Tiempo (segundos)") +
70   ylab("Subscripciones") +
71   theme(legend.position = "bottom",
72         legend.text = element_text(size = 18, face="bold"),
73         legend.title = element_text(size = 18, face="bold"),
74         axis.text.x = element_text(size = 18, face="bold"),
75         axis.text.y = element_text(size = 18, face="bold"),
76         axis.title.x = element_text(size = 18, face="bold"),
77         axis.title.y = element_text(size = 18, face="bold"))
78
79 plot
80
81 print_LaTeX_plot(OUTPUT_PLOT_DIR, plot = plot, width = 10, height = 5)
82
83 # Export
84 workload <- as.integer(workload$subscriptions)
85 write.table(
86   workload,
87   OUTPUT_CSV_DIR,
88   col.names = FALSE,
89   row.names = FALSE,
90   sep = ";")
```

Listing A.4: Programa en R para generar las gráficas del Throughput y del Tiempo de Respuesta.

```
1 library(ggplot2)
2 library(tikzDevice)
3
4 setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
5
6
7 # Function to export plot to LaTeX
8 print_LaTeX_plot <- function(path, plot, width, height) {
9   tikz(filename = path, standAlone = TRUE, width = width, height = height)
10  print(plot)
11  dev.off()
12  tools::texi2pdf(path, quiet = TRUE)
13 }
14 export_plots = TRUE
15
16 ##### Vars. and flag #####
17
18 # Paths of the dir with measurements and of each csv
19 DIR_PATH = "path/to/dir/of/measurements"
20 FILE_NAME_TH = "path/to/th.csv"
21 FILE_NAME_RT = "path/to/rt.csv"
```

```
22
23 # Specify plots
24 PLOT_TH = TRUE
25 PLOT_RT = TRUE
26
27 # Number of points on the plot
28 N_POINTS = 50
29
30 THEME <- theme(legend.position = "bottom",
31                legend.text = element_text(size = 18, face="bold"),
32                legend.title = element_text(size = 18, face="bold"),
33                axis.text.x = element_text(size = 18, face="bold"),
34                axis.text.y = element_text(size = 18, face="bold"),
35                axis.title.x = element_text(size = 18, face="bold"),
36                axis.title.y = element_text(size = 18, face="bold"))
37
38 ##### THROUGHPUT #####
39
40 if (PLOT_TH) {
41
42   # Name of plot labels and generated PDF
43   OUT_FILE_TH = paste("./th_",FILE_NAME_TH,".tex", sep="")
44   XLABEL_TH = "Tiempo (segundos)"
45   YLABEL_TH = "Throughput (notifs/s)"
46
47   data_th <- read.csv(paste(DIR_PATH, FILE_NAME_TH, sep="/"), header = TRUE, sep = ",")
48
49   # Delete first values if needed (system not entirely up, so times are very high)
50   data_th <- data_th[-c(0:3),]
51
52   # Spit data frame in N_POINTS equal-sized data frames and find mean
53   data_th$points <- cut(c(1:nrow(data_th)),
54                         breaks=(nrow(data_th)/N_POINTS)*(0:N_POINTS),
55                         labels=c(1:N_POINTS))
56
57   # Mean for each point
58   new_data_th <- aggregate(cbind(timestamp, throughput) ~ points,
59                         data = data_th, FUN = mean, na.rm = TRUE)
60
61   # To plot all points, replace 'new_data_th' with 'data_th'
62   plot_th <- ggplot(data_th, aes(x = timestamp)) +
63     geom_line(aes(y = throughput), size=1) +
64     xlab(XLABEL_TH) +
65     ylab(YLABEL_TH) +
66     scale_y_log10() +
67     THEME
68
69   print_LaTeX_plot(OUT_FILE_TH, plot = plot_th, width = 10, height = 5)
70 }
71
72 ##### RESPONSE TIME #####
73
74 if (PLOT_RT) {
75
76   # Name of plot labels and generated PDF
77   OUT_FILE_RT = paste("./rt_",FILE_NAME_RT,".tex", sep="")
78   XLABEL_RT = "ID Notificacion"
79   YLABEL_RT = "Tiempo de respuesta (segundos)"
80
81   data_rt <- read.csv(paste(DIR_PATH, FILE_NAME_RT, sep="/"), header = TRUE, sep = ",")
82
83   # Delete first values if needed (system not entirely up, so times are very high)
84   data_rt <- data_rt[-c(0:1),]
85
86   # Spit data frame in N_POINTS equal-sized data frames and find mean
87   data_rt$points <- cut(c(1:nrow(data_rt)),
88                         breaks=(nrow(data_rt)/N_POINTS)*(0:N_POINTS),
89                         labels=c(1:N_POINTS))
90
91   # Mean for each point
```

```
92    new_data_rt <- aggregate(cbind(id, resp_time) ~ points,
93                             data = data_rt,
94                             FUN = mean,
95                             na.rm = TRUE)
96
97    plot_rt <- ggplot(data_rt, aes(x = id)) +
98      geom_line(aes(y = resp_time), size=1) +
99      xlab(XLABEL_RT) +
100     ylab(YLABEL_RT) +
101     THEME
102
103   print_LaTeX_plot(OUT_FILE_RT, plot = plot_rt, width = 10, height = 5)
104 }
```