ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación: Grado en Ingeniería Informática

Título del proyecto:

Estudio de controladores elásticos simples para servicios en la nube

Alumna: Adriana Arregui Roldán
Tutor: José Ramón González de Mendívil
Pamplona, noviembre de 2017

ÍNDICE

OBJETIVOS Y DESARROLLO

En esta memoria se presenta un estudio sobre diferentes controladores de elasticidad para servicios desplegados en la nube. Una de las características más relevantes de los servicios desplegados en la nube es la capacidad de reasignar recursos a un determinado servicio en función de las prestaciones que ofrece. Los servicios tienen ciertos compromisos adquiridos con sus clientes que suelen venir indicados en forma de Objetivos de Nivel de Servicio (SLO), con independencia de que se haya propuesto o no un Acuerdo (formal) de Nivel de Servicio (SLA). Entre los objetivos más importantes se encuentran el tiempo de respuesta (medio) y la disponibilidad del servicio. En este trabajo nos centramos en el objetivo de mantener el tiempo de respuesta del servicio por debajo de un valor máximo indicado por el cliente.

Es relevante indicar que el tiempo de respuesta de un servicio depende en gran medida de los recursos asignados y de la carga de operaciones que recibe dicho servicio. La saturación de un determinado componente de la aplicación produce un efecto muy negativo en el tiempo de respuesta. De acuerdo con estudios realizados, cuando la utilización de un componente sobrepasa el 80% de su capacidad, el tiempo de respuesta de dicho componente crece de una forma exponencial. Una de las técnicas que se emplean para evitar este hecho es la escalabilidad horizontal. Dicha técnica consiste simplemente en duplicar el componente (añadir más instancias del mismo) con el objetivo de repartir la carga de operaciones entre las diferentes instancias. Para poder llevar a cabo esta técnica se dispone de un distribuidor de carga para las distintas instancias y obviamente, se asume que el diseño de la aplicación tolera dicha duplicidad de componentes.

Los controladores de elasticidad que proponemos en este trabajo asumen que los componentes se pueden gestionar con las técnicas de escalabilidad horizontal. En ese caso, lo que es importante, es conocer en cada momento cuál es el número de instancias por componente que son adecuadas para mantener los tiempos de respuesta dentro de los límites establecidos, en función de la carga de entrada que se está recibiendo en dicho momento. Para evitar que se produzcan excesivos incumplimientos en los tiempos de respuesta, es adecuado emplear, además, técnicas predictivas de la carga (y del tiempo de respuesta) puesto que, una decisión de escalado tomada por el controlador tiene un determinado retardo de aplicación hasta que finalmente es instalada la nueva configuración del servicio.

A diferencia de otros trabajos sobre controladores de elasticidad, en este trabajo presentamos técnicas que no emplean la construcción explícita de un modelo analítico del servicio. Se intentan diseñar controladores que únicamente utilizan la información del tiempo de respuesta objetivo, el tiempo de respuesta medido (o estimado) y la carga de operaciones del servicio medida (o estimada), a la hora de tomar una decisión de

escalado. En ese sentido se buscan controladores que sean lo más sencillos posibles y se valora su comportamiento elástico.

Siguiendo el objetivo de este trabajo, en esta memoria presentamos en el Capítulo 1 un recopilatorio de conceptos básicos que explican cómo está construida lo que viene siendo el Cloud o nube, junto a sus servicios, y a lo que elasticidad y escalabilidad sobre esta se refiere. Estos conceptos son fundamentales para comprender el desarrollo del estudio realizado.

En el Capítulo 2 describiremos el modelo de rendimiento, un modelo abierto que distingue diferentes clases de operaciones, en el que se ha basado el estudio con sus respectivas fórmulas y transformaciones, así como un proceso de cálculo de carga a partir de un grafo de comportamiento de cliente.

El Capítulo 3 se ha reservado para la explicación de los controladores elásticos, tanto su función como el inconveniente del retardo que presentan a la hora de aplicar decisiones. Para procurar que estos inconvenientes no influyan en exceso en el estudio, en este capítulo también presentamos el desarrollo de un generador de carga, al cual se le atribuye el primer controlador del trabajo.

En el Capítulo 4 mostramos 3 tipos diferentes de controladores: el primero basado en el algoritmo computacional de Hill Climbing, el segundo denominado agresivo-relajado debido a su agresividad para aumentar componentes, y, por último, un controlador fuzzy, en el que su decisión viene determinada por 25 reglas y 5 conjuntos difusos.

El estudio finalizará con las conclusiones a las que se pueden llegar tras el desarrollo experimental explicado a lo largo del documento.

Capítulo 1

INTRODUCCIÓN A LA ELASTICIDAD DE LA NUBE

1.1 Conceptos básicos.

Como uno de los paradigmas en la Informática moderna, se presenta la computación en la Nube, 'Cloud Computing', que permite ofrecer servicios de computación a través de Internet. Bajo este modelo, todo lo que puede ofrecer un sistema informático se ofrece como un servicio, de tal forma que los usuarios, despreocupados de la gestión de recursos, acceden a servicios a través de la nube. Este tipo de computación presenta importantes ventajas [1]: (i) Prestación de servicio a nivel mundial; (ii) Implementación más rápida con menos riesgos; (iii) Actualizaciones automáticas que no afectan negativamente a los recursos utilizados; (iv) Contribuye al uso eficiente de energía. Por el contrario, el modelo no es ideal y presenta desventajas: (i) Sin acceso a Internet, no tienes disponibilidad de acceder a la aplicación; (ii) La confiabilidad de las aplicaciones depende de los proveedores de servicios en la nube; (iii) Seguridad de la aplicación; (iv) Escalabilidad a largo plazo. Aspectos detallados en [1].

Los servicios disponibles en el cloud se pueden clasificar en:

- **Infraestructura como Servicio (IaaS):** La aplicación existente en el cloud ofrece como su principal funcionalidad una infraestructura basada tanto en máquinas físicas como virtuales y capacidad de la gestión de estas. Este servicio ofrece al usuario la posibilidad de crear nuevas máquinas eligiendo configuraciones de hardware (CPU, memoria, discos duros y capacidad…) o de software (sistema operativo o imagen base para la instancia). Amazon AWS [2], Openstack [3] o Google Compute Engine from Google Cloud [4] son ejemplos de empresas populares que apoyan las IaaS.

- **Plataforma como Servicio (PaaS):** En este caso, el servicio esta ofrecido por una plataforma que ofrece a través de la web un entorno donde poder desarrollar y ejecutar aplicaciones creadas por el usuario. El esfuerzo se realiza únicamente en el desarrollo de la aplicación.Este tipo de plataformas ofrecen sistemas de elasticidad tanto vertical como horizontal (conceptos que se ven en el siguiente apartado) para las aplicaciones que tienen en funcionamiento. Ejemplos de PaaS son Google App Engine [5], Heroku [6] o Azure [7], entre otros.

- **Software como Servicio (SaaS):** El usuario recibe como servicio una aplicación online directamente del proveedor. Bitnami [8], Salesforce [9] o Atlassian Cloud [10] son ejemplos de SaaS.

La siguiente figura [1.1] muestra la organización de la clasificación anterior en capas según la responsabilidad de cada una de ellas. El proveedor del cloud es quien controla

el aprovisionamiento de máquinas virtuales y su asignación a los recursos físicos disponibles; mientras que el proveedor de la aplicación tiene la responsabilidad de asignar componentes de la aplicación sobre la infraestructura proporcionada por el proveedor cloud.

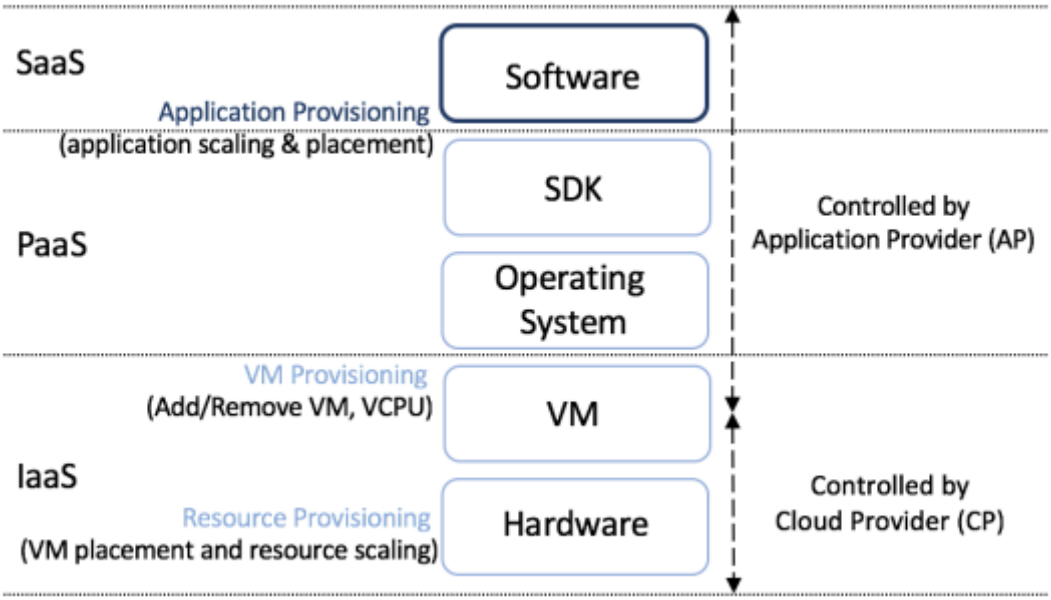


Figura 1.1: Capas con las distintas responsabilidades [19]

1.2 El concepto de la Elasticidad.

En los últimos años se está prestando una importante atención en el desarrollo y despliegue de aplicaciones distribuidas en plataformas Cloud. La creciente demanda de realizar aplicaciones en el Cloud se debe al novedoso modelo 'pay-as-you go', que consiste en pagar por la infraestructura empleada en cada momento.

Una de las ventajas de este servicio ofrecido, es la posibilidad de aprovisionar recursos bajo demanda, según las necesidades de la aplicación. De esta forma los clientes pueden escalar (scale-up or scale-down) los recursos al servicio para mantener la calidad del servicio de la aplicación en todo momento.

De la mano del concepto anterior, aparece el término de elasticidad, el cual se desarrolla a continuación atenuando la visión dada por Herbst [11]:

- **Elasticidad:** grado por el cual un sistema es capaz de adaptarse a los cambios de la carga de trabajo soportados mediante el abastecimiento de recursos de cómputo de forma autónoma; por tanto, en cada punto del tiempo los recursos disponibles encajan con la demanda de recursos real tan cerca como sea posible.

- **Escalabilidad:** un sistema es escalable si sus prestaciones no se degradan considerablemente cuando la carga del sistema aumenta.

Tras estas definiciones, observamos que para que un sistema sea elástico, es requisito indispensable su escalabilidad. La diferencia entre ambos conceptos es que en la escalabilidad el tiempo no tiene ningún papel, por ejemplo, no se tiene en cuenta en qué momento realizar acciones de escalado.

Cabe señalar que para conseguir un sistema escalable es necesario un esfuerzo en el diseño del sistema, es decir, es necesario utilizar diferentes técnicas que permitan que el servicio no se degrade rápidamente en cuanto la carga de trabajo aumente, como por ejemplo técnicas que permitan evitar bloqueos, repartir la ejecución de procesos y datos en diferentes nodos, replicar componentes de una aplicación entre otras.

Debido al auge del uso de Máquinas Virtuales (MVs) y contenedores, existen dos técnicas de escalado que se han hecho populares en los últimos años:

- Escalado horizontal: consiste en la creación o destrucción de componentes de la aplicación con el objetivo de adecuar, de la forma más eficiente posible, la carga de estos, consiguiendo que la aplicación cumpla con el Service Level Agreement (SLA).
- Escalado vertical: consiste en aumentar o reducir los recursos de la máquina que está abasteciendo a la aplicación con el objetivo de cumplir con el SLA.

En este trabajo nos centraremos en el escalado horizontal, que como se puede observar se consigue creando o destruyendo componentes de la aplicación mediante técnicas de replicación y distribución de carga, lo cual realmente supone un cambio en la configuración del sistema, y, por consiguiente, en los recursos que emplea dicho sistema.

Pero para que el escalado horizontal permita convertir al sistema en un sistema elástico debe existir un proceso de adaptación que permita en cada momento tomar la decisión de qué componentes deben duplicarse o reducirse y en qué cantidad.

Se deben evaluar ciertos aspectos de la elasticidad, como son:

- Escalado autónomo: qué proceso de adaptación se utiliza.
- Dimensiones de elasticidad: qué tipo de recursos son escalados en el proceso de adaptación.
- Unidades de los recursos que escalan: para cada tipo de recursos, qué repercusión tiene la variación de cantidad en la asignación.
- Límites de escalado: para cada tipo de recursos, cuáles son las cotas superiores e inferiores en la cantidad de recursos que pueden ser asignados.

Cada tipo de recurso puede verse como una dimensión diferente en el proceso de adaptación con sus propias políticas de elasticidad, aunque normalmente el

abastecimiento de estos recursos es de forma discreta. El sistema funcionando con un mecanismo de elasticidad puede estar en un momento dado sobre abastecido, 'overprovisioned', o, por el contrario, bajo abastecido, 'underprovisioned' en comparación con un valor ideal de abastecimiento óptimo que cumple con la calidad de servicio exigida.

De esta forma, los dos aspectos centrales del mecanismo de adaptación pueden ser:

1. Velocidad: el tiempo que tarda el sistema de pasar de un estado bajo abastecido a un estado óptimo o sobre abastecido, o viceversa.

2. Precisión: la desviación de la cantidad actual de los tipos de recursos asignados respecto a la demanda de recursos óptima en ese momento.

En un sistema real es difícil conocer de antemano los recursos óptimos que se necesitan. Dado que el objetivo de nuestro trabajo es comparar diferentes estrategias de elasticidad, las medidas que se conciliarán en este trabajo serán relativas al coste, a la varianza y a la desviación estándar de este último, al número de incumplimientos del SLA, al número de veces en las que el tiempo está entre los valores de tUp y tDown y al número totales y cambios por iteración que realiza cada controlador. Los resultados de estas medidas nos permiten comparar unos controladores con otros. No obstante, en el capítulo 3 desarrollamos los conceptos de estas medidas.

Capítulo 2

MODELO DEL SISTEMA.

2.1 El modelo de rendimiento.

El diseño de un modelo de rendimiento es una técnica clave para atender los problemas en los sistemas de Tecnología de la Información (TI). Debido a la complejidad que existe al estimar el rendimiento, el diseñador de un servicio debe conocer los límites del sistema a priori, como es el número máximo de operaciones por segundo que el sistema es capaz de procesar o el tiempo mínimo de respuesta que puede alcanzar el sistema.

Los modelos analíticos de rendimiento [12] captan aspectos fundamentales de un sistema informático y los relacionan entre sí mediante fórmulas matemáticas y/o algoritmos computacionales, a partir de una cierta información de entrada, como por ejemplo la carga de trabajo o la demanda de cada componente de la aplicación. Los parámetros de entrada que necesitan los modelos analíticos se pueden dividir en tres categorías [12]:

1. **Parámetros del sistema:** son las características del sistema que afectan al rendimiento, como, por ejemplo, el número de procesos en ejecución, tamaño de buffer o protocolos de red.
2. **Parámetros de los recursos:** son las cualidades intrínsecas de un recurso que influyen en el rendimiento como puede ser el tiempo de búsqueda de discos, latencia y la velocidad de CPU.
3. **Parámetros de carga de trabajo:** se derivan de la caracterización de la carga de trabajo y se pueden subdividir en:
    - *Parámetros de intensidad de la carga de trabajo:* proporcionan una medida de la carga colocada en un sistema, indicada por el número de unidades de trabajo para los recursos del sistema, como son el número de sesiones iniciadas por día o el número de transacciones de Base de Datos (BBDD) ejecutadas por unidad de tiempo.
    - Parámetros de demanda de servicio de carga de trabajo: especifican la cantidad total de tiempo de servicio requerido por cada componente en cada recurso. El tiempo de Entrada/Salida (E/S) total en el servidor de BBDD para una función de consulta o el tiempo de CPU en transacciones en el servidor de aplicaciones, son ejemplos de este tipo de parámetros.

El objetivo crítico de analizar y diseñar los sistemas es garantizar que los objetivos de rendimiento se satisfacen. Los servicios de TI son complejos y pueden ser muy caros. Planificar la capacidad de un sistema con un coste razonable para satisfacer las demandas de los clientes es un aspecto fundamental. Una de las ventajas más importantes de la Nube es que los recursos disponibles son (aparentemente) ilimitados

y pueden reasignarse con facilidad mediante la creación y destrucción de máquinas virtuales asignadas a la aplicación, o incluso empleando creación y destrucción de contenedores dentro de las propias máquinas virtuales. Esta facilidad de los sistemas en la Nube posibilita la realización de una planificación dinámica de la capacidad de un servicio. En ese caso decimos que el servicio es elástico [12].

El diseño de modelos analíticos para servicios elásticos en la Nube es una tarea complicada puesto que muchos parámetros son desconocidos debido a la opacidad del proveedor de la Nube. No obstante, se puede proponer una caracterización de un servicio en la Nube empleando modelos analíticos con la salvedad de que los parámetros empleados no sean utilizados directamente a la hora de desarrollar los controladores elasticidad. En ese caso los parámetros que caracterizan el modelo sólo pueden ser estimados mediante otras técnicas de estimación a partir de las variables observables y medidas del servicio.

En este trabajo empleamos un modelo analítico para emular el comportamiento de un servicio en la Nube que dispone de la posibilidad de elasticidad, en particular, dispone de escalabilidad horizontal. Este modelo del sistema será utilizado en los siguientes capítulos como base para las pruebas de diferentes controladores elásticos.

2.2 Acuerdo del Nivel de Servicio (SLA).

Los clientes de los servicios de TI no suelen preocuparse por métricas como la utilización de CPU o el ancho de banda de la red sino, que tienden a estar más interesados en las métricas relacionadas con la Calidad de los Servicios (QoS) proporcionados por el sistema; o lo que es lo mismo, a un cliente únicamente le importa qué tan bien está funcionando el sistema y si puede llegar a realizar su trabajo a tiempo.

Entre las métricas de rendimiento más importantes se destacan las siguientes:

- El tiempo de respuesta de las operaciones del servicio.
- La disponibilidad del sistema.
- La fiabilidad del sistema.

El SLA se concreta entre la administración de cada organización y los clientes para definir una lista de servicios y atributos de calidad, como el tiempo de respuesta, la disponibilidad, el tiempo de reparación, la confiabilidad y el coste [12].

Definir el SLA conlleva manejar los servicios de TI de varias formas:

- Planificación. Determina cuales son los niveles de servicio.
- Garantía. Supervisando el nivel de servicio se puede garantizar que el servicio cumple los requisitos, y en caso contrario, se pueden identificar problemas cuando no se cumple el nivel de servicio.

La garantía es importante en los sistemas en la Nube puesto que un incumplimiento por parte del proveedor de la Nube puede llevar a una penalización económica.

2.2.1 Tiempo de respuesta

Las personas de todo el mundo cada vez interactúan más con sistemas basados en una computadora, y, evidentemente todo el mundo está de acuerdo en que el tiempo de respuesta es un factor crítico y determinativo para la producción personal, tasas de error y satisfacción. Lógicamente a medida que el tiempo de respuesta es inferior la satisfacción mejora notablemente [13]. Pero de esta forma aparece la incertidumbre sobre ¿Cuáles son los límites que se debe establecer para el tiempo de respuesta? Aunque el estudio sobre los comportamientos hombre-máquina son costosos, la respuesta puede determinarse midiendo el impacto en productividad del usuario y estimando el coste de proporcionar tiempos de respuesta mejorados. Estudios han estimado que [5, 6]:

- En 0.1 segundo el usuario percibe que el sistema reacciona instantáneamente.
- 1.0 segundo es el límite en el que un usuario no cambia su pensamiento, pero nota un cierto retraso por parte del sistema
- A los 10.0 segundos el usuario pierde la atención y la interacción con el sistema se pierde.

El tiempo de respuesta es importante con independencia de la existencia o no de un usuario final. Los servicios creados en la Nube se pueden enlazar para crear servicios más complejos. De esta forma los clientes de un servicio pueden ser a su vez otros servicios. En este trabajo vamos a considerar principalmente la elasticidad del servicio para responder a las necesidades del tiempo de respuesta en función de la carga de operaciones que recibe el sistema

2.2.2 Coste

La simple elección de la capacidad de los servidores en un centro de datos o la elección del número de capas arquitectónicas de un sistema, son ejemplos que afectan tanto al rendimiento como al coste del sistema. Por tanto, hay que ser conscientes que la estimación de coste es un componente esencial tanto a la hora de desarrollar y entregar un sistema como a lo largo de su ciclo de vida. Sin estimaciones de coste, no tiene sentido discutir el nivel de servicio los objetivos de desempeño para un negocio específico, ya que los agentes necesitan calcular costes y estimar beneficios para tomar buenas decisiones.

En nuestro trabajo el coste lo hemos definido como la media de instancias configuradas que hay en cada iteración.

2.3 Modelo abierto multiclase

A continuación, desglosaremos el desarrollo de nuestro modelo de rendimiento, es decir estudiaremos el modelo abierto multiclase. Este modelo presenta unas fórmulas

basadas en leyes matemáticas, que son modificadas por la aproximación de Seidmann cuando componentes de una cierta configuración agrega o elimina instancias.

2.3.1 Múltiples clases de cargas de trabajo

El poder de los modelos de rendimiento se hace evidente cuando se utilizan con fines predictivos. Para ello el modelo es esencial para predecir adecuadamente el rendimiento de un sistema bajo diferentes cargas de trabajo. Estas cargas de trabajo no tienden a ser sólo de una misma clase simple de clientes homogéneos, sino que generalmente un cliente difiere de otro cliente, por lo que los sistemas reales experimentan una amplia variedad de clientes con diferentes usos de los recursos. En cambio, los clientes se agrupan en clases de comportamientos similares, que luego se representan en el modelo como el comportamiento promedio de la clase y la población de clientes de la clase.

A pesar de ser complejo la elección de la abstracción de la carga de trabajo y el número correspondiente de clases, es imprescindible la creación de modelos multiclase para capturar los servicios prioritarios y especiales que requiere cada clase de cargas de trabajo. Así, este tipo de modelos pueden ser utilizados por proveedores de servicios para representar los diferentes requisitos de QoS para las diferentes clases de carga de trabajo y conseguir dimensionar adecuadamente los servidores de la manera más rentable, ya que sintetizar a una sola clase varias clases de cargas de trabajo resulta ser una representación de trabajo inexacta con resultados poco significativos.

Los modelos de una sola clase no pueden responder a las cuestiones de rendimiento importantes relacionadas con clases de carga de trabajo específicas, ya que no pueden distinguir las diferencias entre grupos de operaciones, es decir, los modelos de una sola clase son efectivos en la captura del comportamiento global, pero son limitados en su capacidad predictiva del comportamiento individual del grupo.

No obstante, aunque los modelos de varias clases son más útiles y naturales para describir cargas de trabajo de sistemas reales, presentan problemas al modelador, ya que la precisión del modelo está fuertemente influenciada por el número de clases de carga de trabajo elegidas. Por ejemplo, es difícil obtener parámetros para modelos con varias clases y se deben hacer inferencias para parametrizar cada clase de carga de trabajo y para repartir el sistema entre las clases.

2.3.2 Modelo abierto estacionario

Para desarrollar el concepto de modelo abierto, en el libro Performance by Design [12] hace uso de un servidor de archivos cuyo propósito es proporcionar servicios de archivos para los procesos del cliente en una red de área local de alta velocidad (LAN). Básicamente, un servidor de archivos está compuesto de procesadores, memoria y un subsistema de disco, y su carga de trabajo se puede ver como una serie de solicitudes de servicio de archivos, como lectura, escritura, creación y eliminación, que llegan desde las estaciones de trabajo a través de la LAN. El funcionamiento de un servidor de archivos es: una solicitud típica entra al servidor, posiblemente se encola y espera a que

algún recurso esté disponible, obtiene el recurso del servicio y sale del servidor. El número de solicitudes que el servidor maneja simultáneamente varía con el tiempo dependiendo de factores tales como el número de clientes, la capacidad del sistema, la memoria disponible y la velocidad del procesador.

La razón por la que se utiliza como ejemplo un servidor de archivos como modelo abierto, es que este servidor presenta la característica propia de este tipo de modelado: la variación del número de clientes a lo largo del tiempo, o, en otras palabras, la carga que entra es igual a la que sale. Esta variación se debe a la iniciación, ejecución y terminación de los procesos, y se puede representar en un modelo abierto porque tiene un número ilimitado de clientes.

Las características más importantes del modelo abierto estacionario son [12]:

- La tasa de llegada especifica la intensidad de la carga de trabajo, es decir, la intensidad de la carga de trabajo se representa por el número medio de operaciones que llegan por unidad de tiempo.
- A medida que aumenta la tasa de llegada de operaciones, el número de operaciones tiende a crecer sin límite.
- El rendimiento es un parámetro de entrada, ya que este se define como la velocidad de llegada y se conoce desde un inicio. Esto es debido a que el flujo de entrada en el sistema debe ser igual al flujo de salida del sistema.

Por tanto, en un modelo abierto estacionario la carga de trabajo es el parámetro de entrada y el tiempo de respuesta es el parámetro de salida.

2.3.3 Cuantificación del modelo abierto multiclase

Este apartado se centra en los aspectos cuantitativos de Queuing Networks (QN) e introduce los parámetros de entrada y las medidas de rendimiento que pueden ser obtenidos a partir de dicho modelo.

En primer lugar, destacar que este modelo tiene dos parámetros que se deben especificar previamente. Por un lado, encontramos la intensidad de la carga de trabajo, lo que es lo mismo la velocidad a la que llegan las operaciones, y, por otro lado, nos encontramos la demanda de servicio, que en este caso es el requisito de servicio promedio de un cliente.

Una vez especificados estos parámetros descritos y solucionando ecuaciones simples es posible evaluar el sistema, obteniendo medidas de rendimiento tales como:

- Utilización: porción de tiempo que está ocupado el centro de servicio.
- Tiempo de respuesta: tiempo medio de una operación en el centro de servicio (tiempo en cola y tiempo de servicio).
- Número de operaciones en el centro de servicio (tanto encolados como en servicio).

- Productividad: media de operaciones completadas por unidad de tiempo.

Cabe destacar que, en los modelos basados en un sistema de colas, se aplican cuatro leyes fundamentales [12]:

1. **Ley de utilización:** Tiempo que está ocupado el centro de servicio, es decir, la utilización de un centro de servicio es igual al producto del tiempo medio de servicio por operación en el centro de servicio *i* y el rendimiento medio del centro de servicio *i* en ese recurso.

$$U_i = S_i \times X_i$$

2. **Ley de Little:** Determina que el número medio de operaciones en un sistema es igual al producto del rendimiento medio de ese sistema y el tiempo medio empleado en ese sistema por una solicitud.

$$Q_i = X_i \times R_i$$

3. **Ley del flujo forzado:** Establece que los flujos de todas las partes del sistema deben ser proporcionales entre sí, lo cual proporciona la productividad media del sistema.

$$X_i = V_i \times X_0$$

4. **Ley de demanda de servicios:** Media del tiempo total empleado por una operación en el centro de servicio.

$$D_i = V_i \times S_i = \frac{U_i}{X_0}$$

En la siguiente tabla se indica un resumen de los símbolos utilizados:

$U_i \rightarrow utilización\ media\ del\ centro\ de\ servicio\ i$
$S_i \rightarrow\ tiempo\ medio\ de\ servicio\ por\ operación\ en\ el\ centro\ de\ servicio\ i$
$X_i \rightarrow Productividad\ media\ del\ centro\ de\ servicio\ i$
$V_i \rightarrow nº\ de\ veces\ que\ una\ operación\ visita\ el\ centro\ de\ servicio\ i$
$X_0 \rightarrow Productividad\ del\ sistema$
$D_i \rightarrow Media\ de\ la\ demanda\ del\ centro\ servicio\ i, D_i = V_i \times S_i$
$Q_i \rightarrow Media\ de\ longitud\ de\ cola\ en\ el\ centro\ de\ servicio\ i$
$R_i \rightarrow Media\ de\ tiempo\ de\ respuesta\ por\ operación\ en\ el\ centro\ de\ servicio\ i$
$R_i' \rightarrow Media\ de\ tiempo\ de\ residencia, R_i' = V_i \times R_i$
$R_0 \rightarrow Media\ del\ tiempo\ de\ respuesta\ del\ sistema$
$Z \rightarrow Media\ de\ tiempo$
$N \rightarrow Media\ del\ número\ de\ operaciones\ en\ el\ sistema$

El tipo de aplicaciones o servicios que consideramos en este trabajo son de tipo multicapa. En la siguiente figura [2.1] se observa un modelo de colas con arquitectura en tres capas, compuesto por un servidor Web, un servidor de aplicaciones y un servidor de BBDD. En esta arquitectura, las transacciones se envían al servidor Web, que puede que estar constituido por varios servidores equilibrados de carga. Si la transacción necesita acceder a la base de datos, esta se enviará a la capa de BBDD en la que se procesará y se enviará de vuelta al servidor Web pasando por el servidor de aplicaciones.

Aunque en la teoría de colas se habla de centros de servicios para indicar que un recurso tiene asociada una cola, en este trabajo hablaremos en general de componentes de un servicio, sin preocuparnos de los aspectos concretos de su implementación real. En el modelo abierto solo interesan las demandas de dichos componentes y la carga que soportan ya que como veremos a continuación, el tiempo de respuesta del sistema es únicamente la suma de los tiempos de residencia de los componentes con independencia de la forma en la que están enlazados. Como en la figura [2.1] cada componente tendrá una o varias instancias del mismo gestionado todo por medio de un distribuidor de carga.

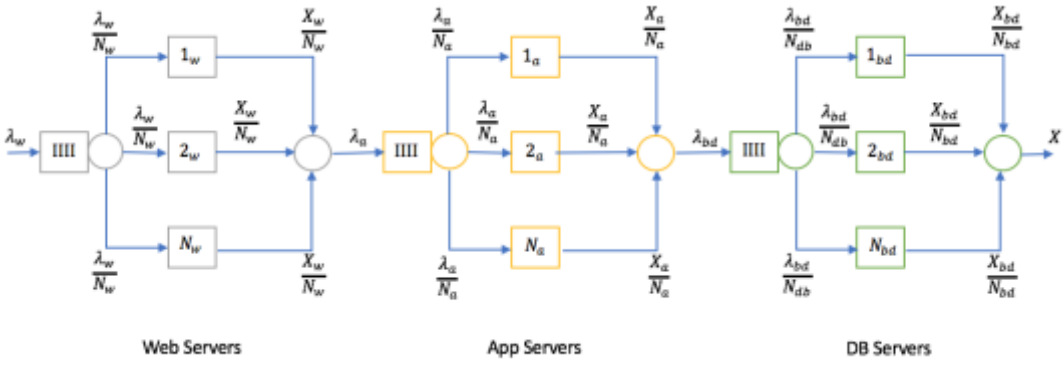


Figura 2.1: Modelo QN de arquitectura en tres capas [16]

Mediante estas cuatro leyes anteriormente descritas, derivan las fórmulas del algoritmo del modelo abierto, como pueden verse a continuación, pero no sin antes recalcar algunas definiciones.

- **Matriz de demandas D:** permite almacenar las demandas para cada componente del sistema por clase de operación. donde las columnas pertenecen a cada clase y las filas a cada centro de servicio.
- **Vector $\vec{N}$:** vector que especifica el número de instancias por componente.
- **Vector $\vec{\lambda}$:** vector que almacena la tasa de llegada para cada clase de operación.
- **Parámetro γ**: parámetro perteneciente a la carga total que llega al sistema.

El siguiente algoritmo pertenece al modelo abierto para únicamente una instancia por componente, $N_i = 1$.

Suponemos que hay C clases $r = 1, \dots, C$, y K componentes, $i = 1, \dots, K$. Por tanto, la matriz D tiene una dimensión $K \times C$, el vector $\vec{N}$ tiene dimensión $1 \times K$, y el vector $\vec{\lambda}$ tiene dimensión 1 x C.

Input: λ_r ratio de llegada por clase, $D_{i,r}$ demandas, γ carga total del sistema.

Output:

Utilización por componente y clase

$$U_{i,r}(\vec{\lambda}) = \lambda_r \times D_{i,r}$$

Utilización total por componente

$$U_i(\vec{\lambda}) = \sum_{r=1}^{C} U_{i,r}(\vec{\lambda})$$

Tiempo de residencia

$$R'_{i,r}(\vec{\lambda}) = \frac{D_{i,r}}{1 - U_i(\vec{\lambda})} = \frac{D_{i,r}}{1 - \sum_{r=1}^{C} \lambda_r \times D_{i,r}}$$

Tiempo de respuesta por clase

$$R_r(\vec{\lambda}) = \sum_{i=1}^{K} R'_{i,r}(\vec{\lambda})$$

Tiempo medio de respuesta

$$\bar{R}(\vec{\lambda}) = \sum_{r=1}^{C} R_r(\vec{\lambda}) \times \frac{\lambda_r}{\gamma}$$

2.3.4 El coste de añadir más instancias. Aproximación de Seidmann

En general, para cada componente del centro de servicio *i* puede haber N_i instancias idénticas entre sí, sobre las cuales un equilibrador de carga perfecto envía una carga de $1/N_i$. Por lo tanto, la tasa media de llegada en cada componente es igual a λ_r/N_i donde λ_r es la tasa de llegada global para la clase r. En la figura [2.2] se puede observar la representación de esta construcción con instancias idénticas.

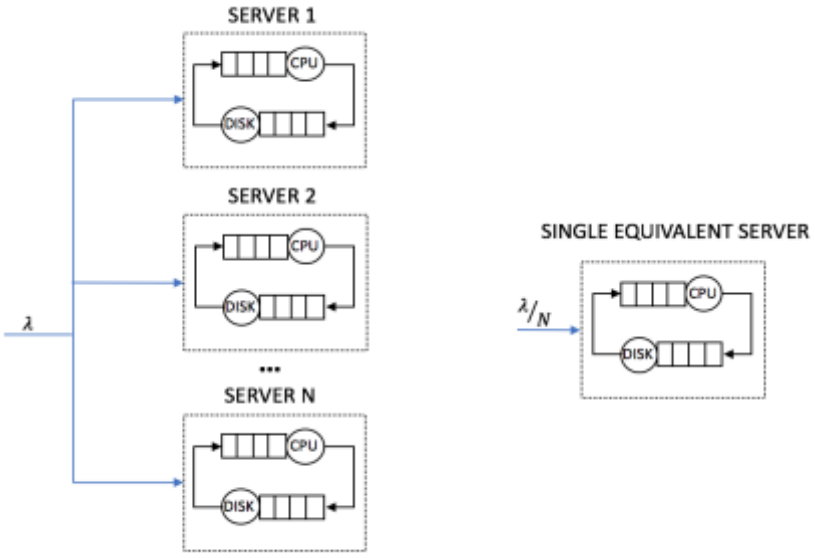


Figura 2.2: Servidor Web simple equivalente a servidor Web múltiple [12]

Así pues, las fórmulas anteriores son modificadas para un componente *i* con N_i instancias idénticas:

Input: λ_r ratio de llegada por clase, $D_{i,r}$ demandas, $\vec{N}$ número de instancias en cada componente, γ carga total del sistema.

Output:

Utilización por componente y clase

$$U_{i,r}(\vec{\lambda}) = \frac{\lambda_r}{N_i} \times D_{i,r}$$

Utilización total por componente

$$U_i(\vec{\lambda}) = \sum_{r=1}^{C} U_{i,r}(\vec{\lambda})$$

Tiempo de residencia

$$R'_{i,r}(\vec{\lambda}) = \frac{D_{i,r}}{1 - U_i(\vec{\lambda})} = \frac{D_{i,r}}{1 - \sum_{i=1}^{C} \frac{\lambda_r}{N_i} \times D_{i,r}}$$

Tiempo de respuesta por clase

$$R_r(\vec{\lambda}) = \sum_{i=1}^{K} R'_{i,r}(\vec{\lambda})$$

Tiempo medio de respuesta

$$\bar{R}(\vec{\lambda}) = \sum_{r=1}^{C} R_r(\vec{\lambda}) \times \frac{\lambda_r}{\gamma}$$

Los autores [16] utilizan la aproximación de Seidmann para reemplazar un componente múltiple con varias instancias por un componente independiente de la carga, con demandas bien ajustadas, y un dispositivo de retardo, que simboliza el empleo de tiempo en la gestión de instancias. Esto último se puede representar como en la siguiente figura [2.3]:

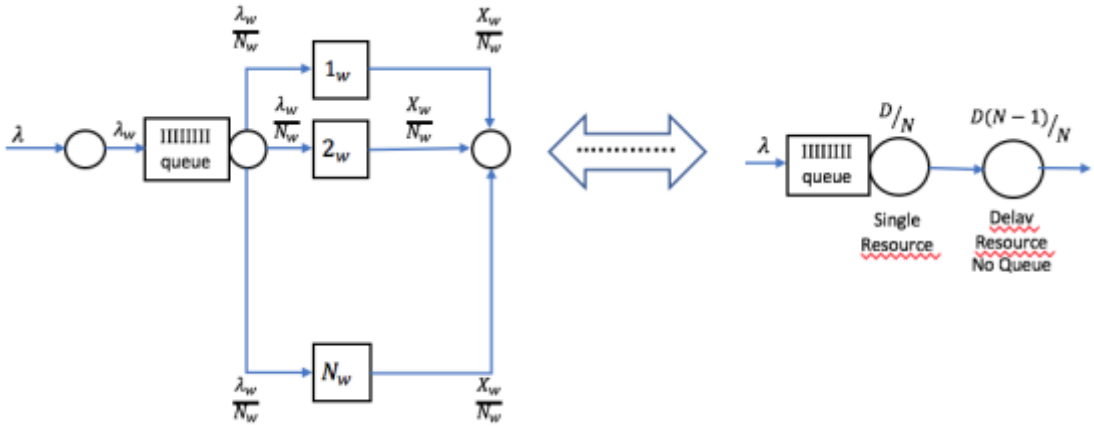


Figura 2.3: Aproximación de Seidmann [16]

Como es obvio, la penalización debe ser mayor conforme más instancias se tiene en el componente. Así pues, el tiempo de residencia para el modelo abierto sería:

$$R'_{i,r}(\vec{\lambda}) = \frac{D_{i,r}}{1 - U_i(\vec{\lambda})} + \frac{D_{i,r}}{N_i} \times (N_i - 1)$$

Cabe destacar que en lugar de utilizar la aproximación de Seidmann, se podría utilizar un sistema de una sola cola con tasas de servicio dependientes de la carga, pero la aproximación descrita permite que el sistema sea más manejable.

2.4 MODELO DEL SISTEMA UTILIZADO EN LAS PRUEBAS

Nuestro modelo de sistema está basado en un sistema web, con arquitectura multicapa, al que se le atribuye un grafo para representar el comportamiento de un cliente (CBMG) durante una sesión. Sesión se refiere a la secuencia de solicitudes consecutivas procedentes de un mismo cliente al servicio. Un grafo CBMG tiene un nodo para cada estado en el que puede estar el cliente durante una sesión, es decir, cada clase de operación que puede desarrollar el cliente. Estas clases de operación están unidas mediante arcos que indican la probabilidad de transiciones entre unas y otras. El modelo que presentamos en este trabajo está constituido por 6 componentes y cada componente puede llegar a tener un máximo de 10 instancias y un máximo de 6 clases distintas de operaciones.

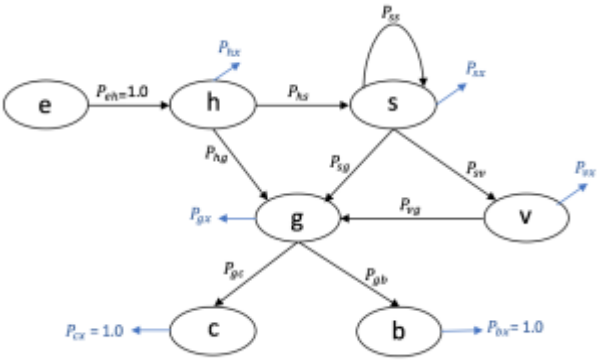


Figura 2.4: Grafo de comportamiento de un cliente (CBMG) [12]

En la figura anterior [2.4] observamos el grafo que utilizaremos para desarrollar nuestro estudio. La clase de operación (e) representa al cliente justo antes de entrar al sitio. Los arcos representan la probabilidad de transición directa de la clase *i* a la clase *j*. La clase (*x*), representada con flechas azules, es el estado al que el cliente puede acceder desde cualquier estado *i* con probabilidad p_{ix}.

Es importante tener presente el grafo de comportamiento que representa a cualquier tipo de aplicación Web ya que, a partir de éste, se obtiene la carga de trabajo para cada tipo de clase de operación. Este proceso comienza obteniendo las visitas a cada estado realizadas por un cliente en una sesión. Según el grafo del ejemplo, el sistema lineal de ecuaciones a resolver para la obtención de las visitas sería:

$$
\begin{aligned}
V_e &= 1 \\
V_h &= V_e \times p_{eh} = 1 \\
V_s &= V_h \times p_{hs} + V_s \times p_{ss} \\
V_v &= V_s \times p_{sv} \\
V_g &= V_h \times p_{hg} + V_s \times p_{sg} + V_v \times p_{vg} \\
V_c &= V_g \times p_{gc} \\
V_b &= V_g \times p_{gb}
\end{aligned}
$$

Para la resolución del sistema lineal es necesario tener una matriz de probabilidad o de transición. Esta matriz de transición tendrá un tamaño de *r* x *r*, donde *r* es el número de clases de operaciones que admite el sistema, y la que plasmará por cada fila la probabilidad de cambio de estado para cada columna o no. En otras palabras, plasmará los arcos del CBMG para cada uno de los clientes que den uso a la aplicación.

Una vez resuelto el sistema lineal para dos matrices de transición de dos sesiones (dos clientes diferentes de la aplicación), con los resultados de las visitas y la definición de γ como la carga total que entra al sistema y sean f_A y f_B la fracción de sesión del tipo A y B respectivamente, la carga para cada una de las anteriores clases sería:

$$
\begin{aligned}
\lambda_h &= \gamma(f_A \times V_h^A + f_B \times V_h^B) \\
\lambda_s &= \gamma(f_A \times V_s^A + f_B \times V_s^B) \\
\lambda_v &= \gamma(f_A \times V_v^A + f_B \times V_v^B) \\
\lambda_l &= \gamma\left(f_A \times V_g^A + f_B \times V_g^B\right)
\end{aligned}
$$

$$\lambda_c = \gamma(f_A \times V_c^A + f_B \times V_c^B)$$
$$\lambda_b = \gamma(f_A \times V_b^A + f_B \times V_b^B)$$

Con todos los valores obtenidos hasta ahora junto a la matriz de demandas del sistema podemos aplicar las fórmulas que describen el modelo de rendimiento descrito.

2.4.1 Configuración del modelo

A partir de las matrices de probabilidad, tomadas todos ellos del libro de Menascé [12], en las que existen 6 clases de operaciones, en nuestro estudio hemos obtenido resultados similares a los del libro mencionado, como se observa a continuación:

- MatA = $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0 & 0.1 & 0 & 0 & 0.2 \\ 0 & 0 & 0.4 & 0.2 & 0.15 & 0 & 0 & 0.25 \\ 0 & 0 & 0 & 0 & 0.65 & 0 & 0 & 0.35 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & 0.6 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ → Visitas A = $\begin{pmatrix} 1 \\ 1 \\ 1.167 \\ 0.233 \\ 0.427 \\ 0.128 \\ 0.256 \end{pmatrix}$

- MatB = $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.7 & 0 & 0.1 & 0 & 0 & 0.2 \\ 0 & 0 & 0.45 & 0.15 & 0.1 & 0 & 0 & 0.3 \\ 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0.6 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & 0.55 & 0.15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ → Visitas B = $\begin{pmatrix} 1 \\ 1 \\ 1.273 \\ 0.191 \\ 0.304 \\ 0.091 \\ 0.167 \end{pmatrix}$

Fijando de forma aleatoria una carga total máxima $\gamma = 50$ y unas frecuencias para A y B, $f_A = 0.15$ y $f_B = 0.85$, hemos obtenido como vector $\vec{\lambda}$:

- $\vec{\lambda} = (50 \;\; 62.8409 \;\; 9.8636 \;\; 16.1015 \;\; 4.8314 \;\; 9.0175)$

Para los cálculos de nuestro modelo es necesario tener una matriz de demandas, que representa cuanto está de demandado un componente por cada clase del sistema. Esta matriz es la siguiente:

- $D = \begin{pmatrix} 0.008 & 0.009 & 0.011 & 0.06 & 0.012 & 0.015 \\ 0.03 & 0.01 & 0.01 & 0.01 & 0.01 & 0.01 \\ 0 & 0.03 & 0.35 & 0.025 & 0.045 & 0.04 \\ 0 & 0.008 & 0.08 & 0.009 & 0.011 & 0.012 \\ 0 & 0.01 & 0.009 & 0.015 & 0.07 & 0.045 \\ 0 & 0.035 & 0.018 & 0.05 & 0.08 & 0.09 \end{pmatrix}$

En esta matriz de demandas cada $D_{i,r}$ representa la demanda solicitada para el componente i por una operación de clase r.

2.4.2 Carga – Respuesta

No obstante, si iteráramos el desarrollo anterior con una configuración de sólo una instancia por cada uno de los 6 componentes que determinamos para nuestro estudio, la capacidad de los componentes se saturaría y, por consiguiente, el sistema dejaría de funcionar. Para ello estudiamos el máximo tiempo de respuesta medio en función de la carga restringiendo la utilización de los componentes.

En este apartado para comprender el funcionamiento del sistema vamos a estudiar el comportamiento del mismo con diferentes configuraciones. Las configuraciones se fijan a $\vec{N} = [1, \dots, 1]$ hasta $\vec{N} = [10, \dots, 10]$ con el mismo número de instancias por componente. La idea es obtener la carga máxima que soporta el sistema con esa configuración con un máximo de utilización en cada componente. El resultado obtenido fue la siguiente gráfica:

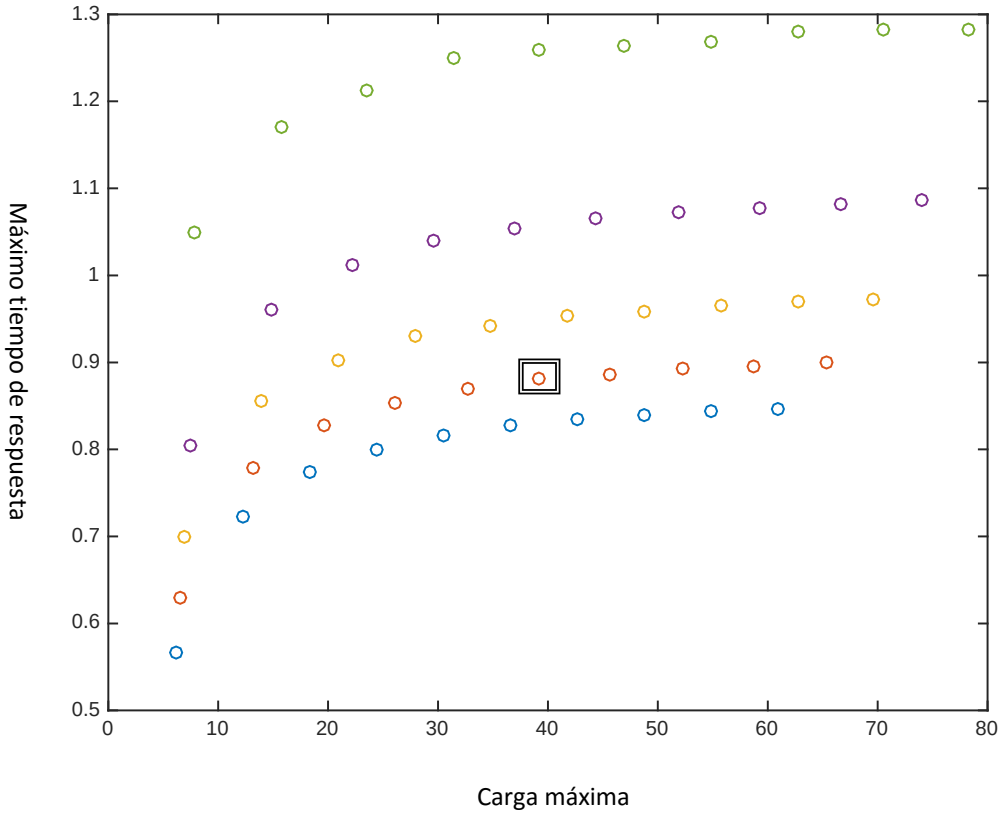


Figura 2.5 Gráfica carga-respuesta

Siendo el color azul el valor de 70% de utilización y el verde 90%, en esta gráfica podemos observar que, por ejemplo, el punto que está dentro del cuadro, a una utilización máxima de 75% y con una configuración de 6 instancias por componente (36 componentes en el sistema), el sistema puede soportar una carga máxima de 40 con un tiempo medio de casi 0.9 segundos.

Con este estudio podemos determinar los valores de los parámetros necesarios. Para ello, previamente escogemos la utilización de los componentes a la que parece que el sistema puede rendir de la forma más óptima, utilización a la que parece que el tiempo de respuesta aun no aumenta de forma exponencial. Nosotros hemos escogido una utilización media, 80% (color amarillo en la figura 2.5). Una vez escogida la utilización, determinamos los siguientes valores basándonos en los resultados de configuraciones medias:

- $T_{SLA} = 1.2$
- $T_{up} = 0.96$
- $T_{down} = 0.7$

Los valores de estos parámetros son importantes para nuestro estudio por lo que su cálculo debe ser muy específico. En el siguiente capítulo veremos el significado de estos parámetros y su importancia.

Capítulo 3

CONTROLADORES Y MEDIDAS

3.1 Controladores elásticos

Haciendo un repaso de lo descrito hasta aquí, se presenta la siguiente figura [3.1] en la que se observa que, a partir de una carga, calculada mediante un grafo de comportamiento de clientes de una aplicación Web multicapa, que entra a un sistema, el cual hemos descrito nosotros por el estudio de carga respuesta, se obtiene el tiempo de respuesta para dicha carga de entrada

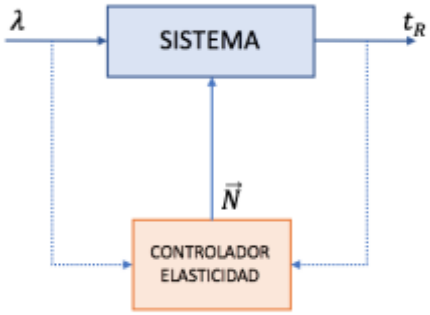


Figura 3.1 Controlador elástico

Puede suceder que la configuración del sistema no consiga tener una buena calidad de servicio en cuanto al tiempo de respuesta. Para ello se incorpora un controlador de elasticidad, el cual a partir de la carga que entra y el tiempo de respuesta obtenido, calcula una nueva configuración para conseguir el tiempo de respuesta deseado. El uso de un controlador elástico tiene un problema. Este problema surge porque el cambio de configuración tiene un retardo en su aplicación y puede no ser instalado para cargas que necesitan esa nueva configuración.

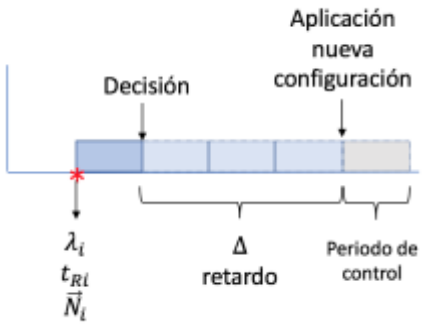


Figura 3.2: Retardo de instalación

Observamos en la figura que desde que se toma la decisión hasta que se aplica la configuración decidida, existe unos periodos de retardo. El retardo en la aplicación de una nueva configuración conduce a que ciertos periodos estén fuera de control y que en esos periodos se degrade el comportamiento del sistema, pudiendo obtenerse tiempos de respuesta por encima del tiempo SLA establecido.

Existen dos tipos de controladores elásticos:

-Reactivo: Es el proceso anterior, es decir, calcula el tiempo de respuesta del sistema a partir de una carga de entrada y con ello calcula una nueva configuración para la siguiente entrada de carga.

-Proactivo: En este caso el controlador estima una carga de entrada y con ello un tiempo de respuesta (estimado), lo que le ayuda al sistema a calcular una configuración para una carga estimada que llegará en siguientes iteraciones.

El controlador reactivo no resuelve claramente el problema anterior que es aplicar una nueva configuración a tiempo cuando es necesaria. Por consecuencia, se decidió desarrollar el proyecto con un controlador elástico proactivo, ya que, si la estimación se realiza para iteraciones muy adelantadas, la configuración puede llegar a ser instalada para esas iteraciones estimadas. Así pues, el proceso de un controlador elástico proactivo sería como el indicado en la figura 3.3.

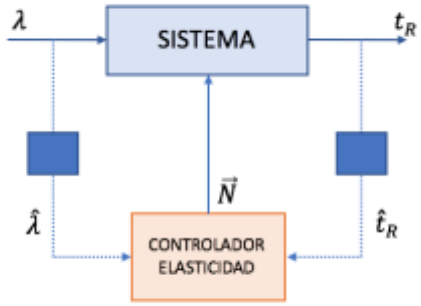


Figura 3.3: Controlador elástico proactivo

Para la estimación de la carga es necesario tener un generador de carga que se aproxime a la realidad como veremos más adelante. De forma aleatoria, hemos determinado que la estimación sea para la iteración *i* + 3, donde *i* es la iteración actual y el tiempo de retardo es de 3 periodos de control, así se instalará la nueva configuración a tiempo para los valores estimados de la carga.

3.2 Parámetros

Mediante la prueba de carga-respuesta, anteriormente descrita en el apartado 4 del Capítulo 2, y fijándonos una utilización del 80% determinamos los siguientes parámetros necesarios para nuestro sistema:

- t_{SLA}: Ya comentado anteriormente, se trata del acuerdo de nivel de servicio. Nosotros determinamos con el estudio de carga-respuesta un valor de tiempo SLA de 1.2 segundos, todo tiempo de respuesta por encima define al servicio con una calidad baja.
- t_{up}: Si el t_{SLA} es inviolable, definimos este valor para poder prevenir que el sistema no lo viole. Si el tiempo de respuesta supera este valor, diremos que el sistema esta infra dimensionado y es necesario añadir más instancias en la configuración. Su valor debe ser algo más bajo que el del t_{SLA} y por esa razón lo determinamos a 0.96.
- t_{down}: Permite determinar que el sistema no esté sobre dimensionado, es decir que no se contengan más instancias de las necesarias. Cuando el tiempo de respuesta del sistema está por debajo del valor 0.7, valor que hemos determinado tras el estudio carga-respuesta, es extremadamente rápido por la sobredimensión del momento, y se deben quitar instancias para mantener el tiempo entre el t_{up} y el t_{down}.
- Como carga máxima que puede llegar a soportar el sistema determinamos 50 operaciones por segundo.

3.3 Medidas

En este trabajo presentamos varios controladores de elasticidad. Con el objetivo de comparar distintos controladores hemos tomado las siguientes medidas:

- Utilidad: Mide cuan útil es la configuración del momento; su cálculo se rige por la fórmula:

$$U = w_1 \times \frac{\lambda}{\gamma} + w_2 \times \left(1 - \frac{\sum_{i=1}^{K} N_i}{maxInstances}\right) + w_3 \times (1 - H)$$

  Donde w_1, w_2 y w_3, correspondiente a cada uno de los 3 términos de la función, es la importancia dada a la carga, a la configuración y al tiempo de respuesta respectivamente, y se cumple: $w_1 + w_2 + w_3 = 1$. En nuestro caso, los valores de estos quedan determinados de forma aleatoria cumpliendo que lo más importante es el tiempo de respuesta y por ello se debe otorgar un peso mayor en la función de utilidad. Aunque se probaron varios valores estos valores quedaron definidos como $w_1 = 0.1, w_2 = 0.4\ y\ w_3 = 0.5$, otorgando más utilidad al tiempo de respuesta y menos a la carga de entrada. El parámetro H corresponde a la función:

$$H \begin{cases} 1 \; si \; rt > \; t_{SLA} \\ \dfrac{rt - \; t_{up}}{t_{SLA} - \; t_{up}} \; si \; t_{up} \leq rt \; < \; t_{SLA} \\ 0 \; si \; t_{down} < rt < t_{up} \\ 0.25 \times \left(\dfrac{-rt}{t_{down}}\right) + 0.25 \;\; si \; rt \leq t_{down} \end{cases}$$

Esta función, donde rt es el tiempo de respuesta, queda representada por la gráfica de la figura 3.4. Como se puede observar claramente, la función H penaliza los tiempos que igualan y superan el tiempo SLA haciendo que el tercer término de la función U, perteneciente al tiempo de respuesta, sea 0. No obstante, los tiempos demasiado rápidos (por debajo del tiempo down) y los tiempos que comienzan a ser lentos (entre el tiempo up y el tiempo SLA) también son penalizados, aunque a menor escala. En esta función salen beneficiados los tiempos aceptables que se encuentran entre el tiempo down y el tiempo up, los cuales aportarán el máximo valor del tercer término de la función U.

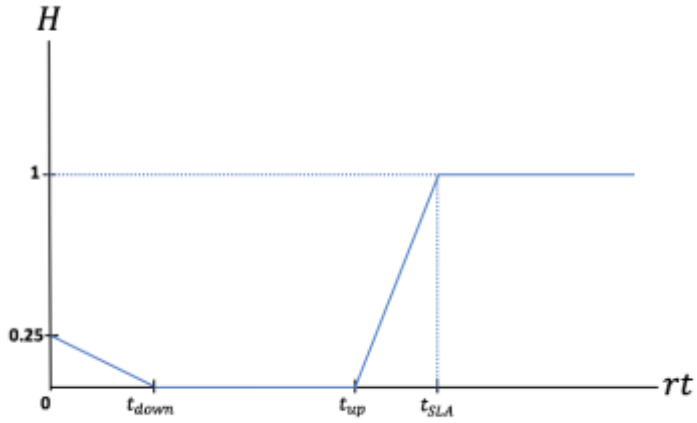


Figura 3.4: Gráfica correspondiente a función H

Finalmente, las utilidades quedan representadas en una gráfica de la figura [3.4] en que se pueden observar la utilidad de los cambios iteración a iteración. Como valores aceptables de utilidad escogeremos utilidades entre el 0.7 y 0.9.

- Número de incumplimientos del SLA: Contamos el número de veces en las que el tiempo de respuesta del sistema supera $t_{SLA} = 1.2$, es decir el número de veces que el sistema está siendo demasiado lento y debe abastecerse con más instancias.

- Número de veces que el tiempo de respuesta se encuentra entre el t_{up} y el t_{down}: parámetro que indica la cantidad de veces en las que el tiempo de respuesta se encuentra en el rango “aceptable” de tiempo.

- Coste medio: Suma del total de instancias en cada iteración y se divide por el número total de iteraciones, que en nuestro caso han sido 3500 iteraciones.

- Varianza del coste medio y desviación típica.

- Media de cambios realizados por iteración: Calculamos el número de veces que añadimos o eliminamos instancias respecto a la configuración anterior y se divide por el número total de iteraciones.

- Gráfica de instancias en cada iteración: Se trata de la representación gráfica del número de instancias que hay en el sistema en cada una de las iteraciones, así se observa como suben y bajan las instancias por cada cambio de carga y tiempo de respuesta.

- Gráfica tiempo de respuesta: Representación visual del tiempo de respuesta calculado en cada iteración, en la cual se puede observar cómo se regula según el número de instancias totales en el sistema.

3.4 Generador de carga

Como se ha comentado anteriormente, para nuestro estudio desarrollamos una función (cargaseno.m) que, a partir de un periodo dado, crea un generador de carga sinusoidal con valores positivos.

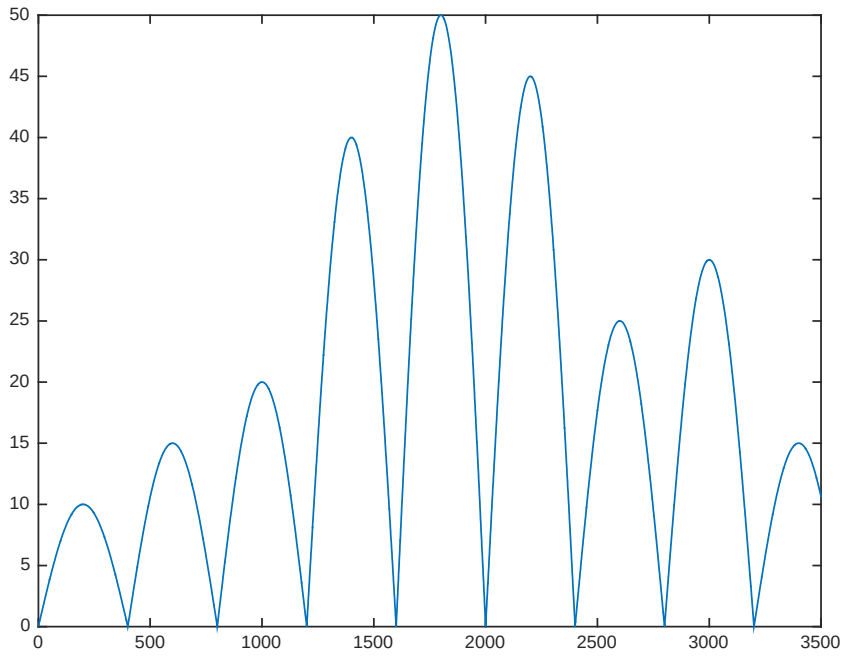


Figura 3.5: Generador de carga sinusoidal

Como podemos observar en la figura 3.5, fijamos un periodo de 400 y para la amplitud, modificable también, los valores fueron 0.2, 0.3, 0.4, 0.8, 1, 0.9, 0.5, 0.6, 0.3, sobre la carga de trabajo máxima que acepta el sistema, 50 en nuestro caso. Así pues, conseguimos un generador de carga creciente o decreciente en cada iteración que recorre todos los valores de carga en 3500 iteraciones.

Desarrollado este generador de carga, se puede explicar que, para facilitar nuestro estudio, el cálculo de la estimación para la carga y por consiguiente el tiempo de respuesta estimado es ideal, ya que en este controlador tiene los valores fijados de antemano y no existe ningún tipo de ruido en la carga. Obviamente esto queda muy lejos de la realidad porque la carga que entra a un sistema puede variar entre iteraciones de forma aleatoria.

Nuestro objetivo es comparar los diferentes controladores de elasticidad en las mismas condiciones ideales, una estimación perfecta y la misma carga. Esto es así puesto que nos interesa analizar la viabilidad de las soluciones. En otras palabras, si no tienen un buen comportamiento en condiciones ideales no podemos aplicarlo en condiciones reales.

3.5 Controlador basado en la carga-respuesta

Presentado el estudio de carga-respuesta y el generador de carga, podemos desarrollar el primer controlador elástico proactivo de forma sencilla. El funcionamiento de este controlador se basa en estimar por parte del sistema una carga de trabajo, $\hat{\lambda}$, enviársela al controlador y que éste configure $\vec{N}$ según entre que rango de valores se encuentre esa carga.

La implementación de este controlador es sencilla:

$$if\ \ \lambda_1 \leq\ \widehat{\lambda} \leq\ \lambda_2\ then$$
$$\vec{N} \leftarrow \text{instancias correspondientes en el rango } \lambda_1\lambda_2$$
$$end$$

Para determinar los valores de los diferentes rangos hemos utilizado los resultados del estudio de carga-respuesta (apartado 4 del Capítulo 2). Como se ha comentado anteriormente, además de determinar los tiempos SLA, up y down, determinamos los valores de los rangos de una configuración u otra a una utilización de 80% (color amarillo en la figura 2.5). Lógicamente los rangos vendrán determinados por los valores entre las marcas circulares que se observan en la gráfica.

Los resultados de las medidas mencionadas para este controlador son:

- Configuración por iteración

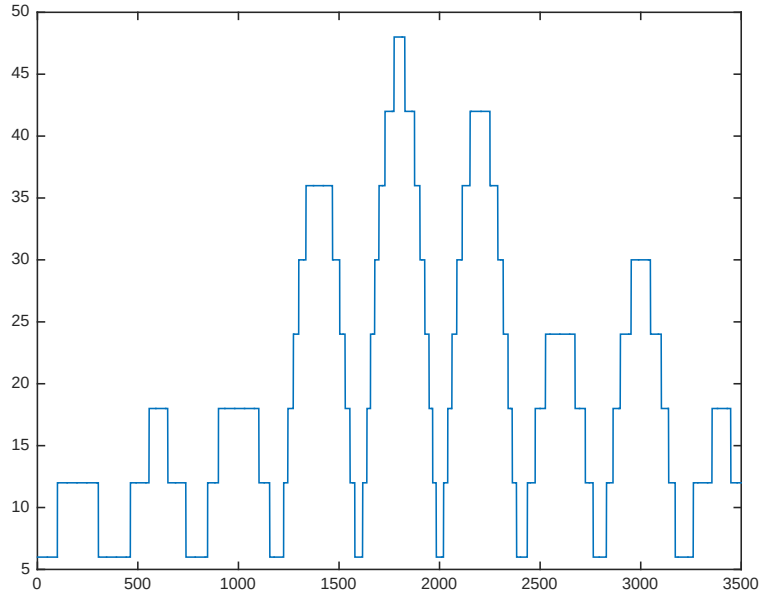


Figura 3.6: Gráfica del abastecimiento de instancias (controlador carga estimada)

- Tiempo de respuesta

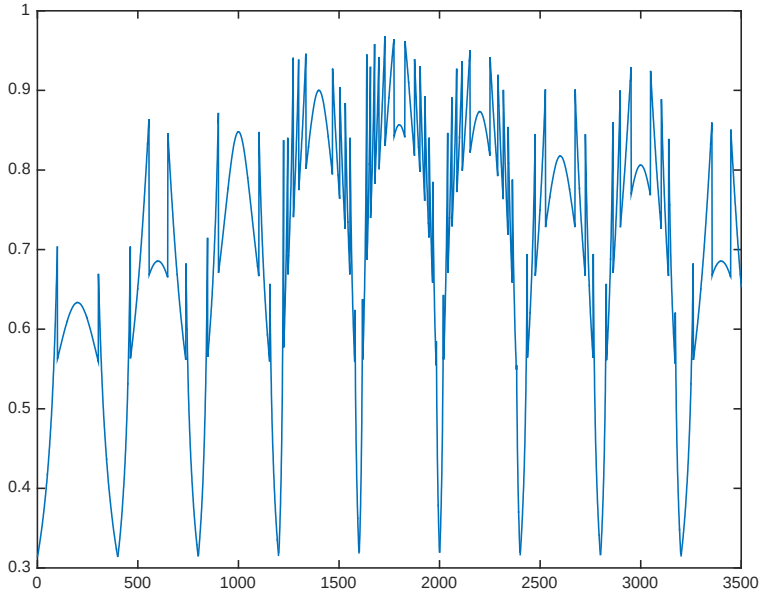


Figura 3.7: Tiempo de respuesta medido (controlador carga estimada)

- Utilidad de los cambios realizados

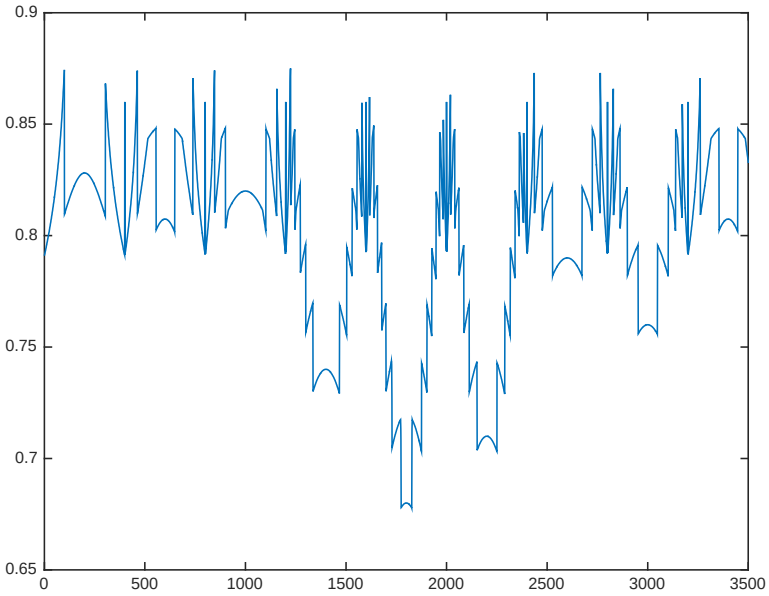


Figura 3.8: Utilidad de los cambios realizados (controlador carga estimada)

Coste medio	18.7217
Varianza / desviación estándar	123.7968 / $\pm$ 11.1264
Nº incumplimiento SLA	0
Between tUP and tDown	1867
Nº cambios de configuración	378
Nº cambios por iteración	0.1080

Este controlador lo hemos desarrollado por la sencillez de su implementación y es el cual nos va a servir de base de mejora. Se trata de un controlador, como hemos comentado, basado en condiciones ideales, con el que se consiguen buenos resultados, ya que no incumple en ningún momento el tiempo SLA y tiene un coste bajo. Además, consigue que en más de la mitad de las iteraciones el tiempo se encuentre entre los valores de tUp y tDown. Aunque para este controlador no todo son ventajas, ya que tiene la gran desventaja de configurar a $\vec{N}$ de forma simétrica, es decir, que todos los componentes tienen el mismo número de instancias. Esto es factor de mejora de nuestro estudio, ya que es posible encontrar configuraciones más variadas y así rebajar el coste, sin abastecer al sistema demás.

Capítulo 4

CONTROLADORES ELÁSTICOS

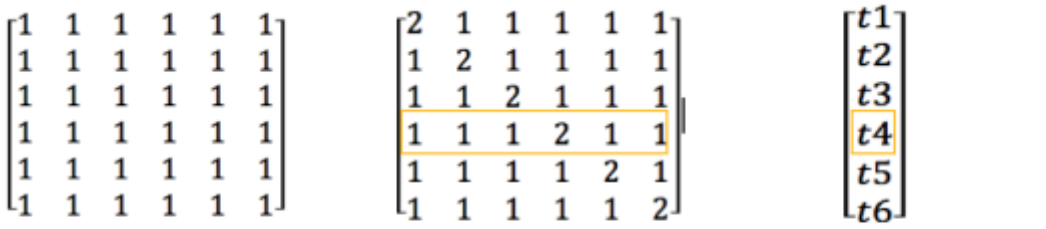
Como se ha comentado, el primer controlador desarrollado en el capítulo anterior de forma tan sencilla, funcionaría a la perfección a un bajo coste y sin incumplimientos del SLA, pero nuestro estudio ha estado enfocado en medir diferentes controladores que funcionen a este nivel o mayor, ya que este controlador realmente sube una instancia a cada uno de los componentes, pero igual se podría conseguir los mismos resultados añadiendo instancias únicamente a ciertos componentes, lo cual rebajaría el coste.

En este capítulo, presentamos los diferentes controladores que hemos implementado, estudiado y comparado, junto a sus medidas.

4.1 Hill Climbing

Este controlador está basado en el algoritmo computacional de Hill Climbing, el cual desde una situación inicial, intenta encontrar una mejor solución variando incrementalmente un único elemento de la solución. Si el cambio produce una mejor solución, otro cambio incremental se le realiza a la nueva solución, repitiendo este proceso hasta que no se puedan encontrar mejoras [17].

En nuestro caso, este controlador se basa en una matriz de tamaño *k* x *k*, siendo *k* el número de componentes existentes en el sistema. Esta matriz se inicializa con una instancia para cualquier componente. A partir de ella, añadimos una segunda instancia al primer componente y dejando el resto de componente a 1, se mide entonces el tiempo de respuesta para esa configuración para la carga estimada, y se almacena. Tras ello, al segundo componente añadimos una segunda instancia y el resto a 1, medimos y almacenamos el tiempo de respuesta, así sucesivamente para todos los componentes. Una vez tenemos los tiempos para todas las configuraciones, escogemos la mejor, es decir, aquella en el que tiempo esté entre el tDown y el tUp. Esa es la configuración para la carga estimada; en los peores casos, si el mejor tiempo supera el tUp el algoritmo repite el proceso inicializando la matriz a la configuración perteneciente a ese mejor tiempo, hasta encontrar el tiempo deseado o hasta alcanzar las instancias máximas.



- Si t4 es un tiempo que se encuentra entre el tUp y tDown, la configuración necesaria para la carga del sistema sería:

$$1 \quad 1 \quad 1 \quad 2 \quad 1 \quad 1$$

- Si t4 se encuentra por encima del tUp y es el mayor de todos, el siguiente paso sería:

$$\begin{bmatrix} 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 2 & 1 & 1 & 2 & 1 & 1 \\ 1 & 2 & 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 2 & 1 \\ 1 & 1 & 1 & 2 & 1 & 2 \end{bmatrix} \quad \begin{bmatrix} t1' \\ t2' \\ t3' \\ t4' \\ t5' \\ t6' \end{bmatrix}$$

Para entrar en este proceso de Hill Climbing, se debe cumplir la condición que si la carga estimada, $\hat{\lambda}$, supera al a la carga máxima para que la configuración tenga una única instancia por componente utilizando los rangos del anterior controlador determinados por el estudio de carga-respuesta. Este proceso se repetiría hasta conseguir que algún tiempo estuviera entre el tUp y el tDown o hasta alcanzar el número máximo de instancias.

Este controlador no se puede implementar en su totalidad porque no se conoce el modelo, pero se incluye para compararlo con los otros controladores.

Para nuestro estudio, el algoritmo de este controlador se ha implementado de la forma:

Crear matriz vecindad de unos de tamaño i x i
Initialize i ← 1, cont ← 1

while $\hat{t}_R > tUp$
    if vecindad (i, cont) + 1 ≤ número máximo de instancias por componente then
        añadir una instancia a esa posición en la matriz
        almacenar t_R con esa nueva configuración
    else
        añadir una posición en las columnas
        if posición supera el número de columnas then
            reiniciar esa posición
        end
        while vecindad (i, posicion) + 1 > que el número máximo de instancias
            añadimos 1 a posición
            if posición supera el número de columnas then
                reiniciar esa posición
            end
            if recorre todas las columnas then
                llevamos un control de los breaks realizados
                break;
            end
        end
        if no ha recorrido todas las posiciones then
            añadir una instancia
            almacenar t_R con esa nueva configuración
        else
            almacenar t_R con la configuración inicial
        end

    end

    if se han recorrido todas las filas de vecindad then
        if existen $t_R almacenados > tUp$ then
            escoger como mejor el mayor de los tiempos
            almacenar la configuración correspondiente a ese valor de tiempo
        else
            escoger como mejor el tiempo más cercano al tiempo medio entre tUp y tDown
            almacenar la configuración correspondiente a ese valor de tiempo
        end

        Volcar la nueva configuración en todas las filas de matriz vecindad
    End

    if se ha recorrido la matriz vecindad then
        Reiniciar cont e i
    end
    sumar uno a cont y a i
    if reliza tantos breaks como número máximo de filas then
      break para salir del while, estamos en instancias máximas
    end

La aplicación de este algoritmo de Hill Climbing ha resultado:

- Configuración por iteración

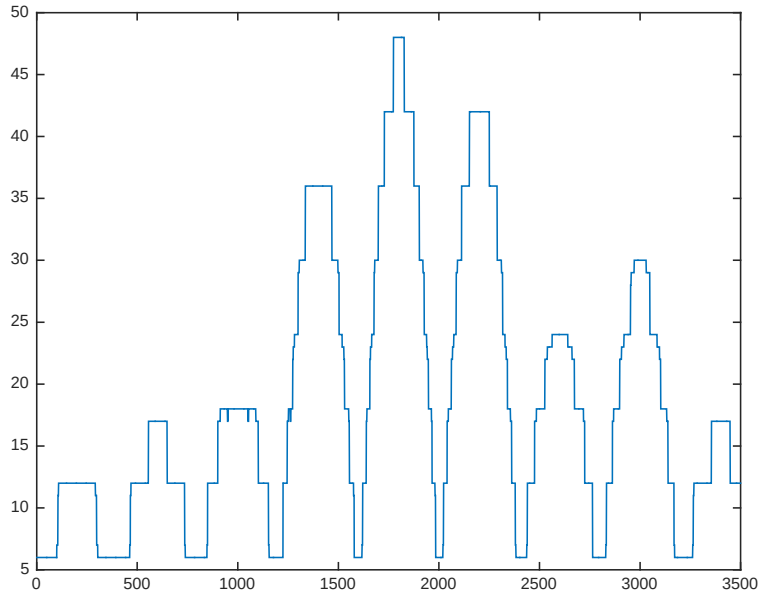


Figura 4.1: Gráfica del abastecimiento de instancias (controlador Hill Climbing)

- Tiempo de respuesta

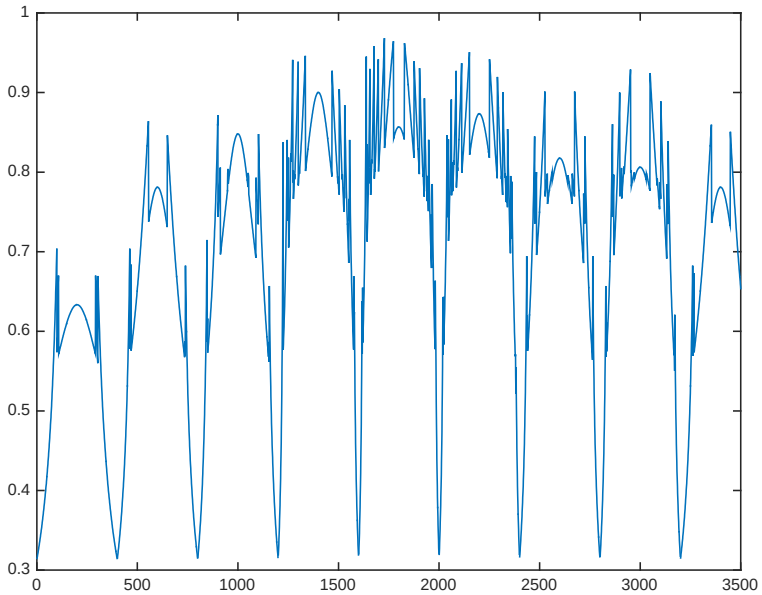


Figura 4.2: Tiempo de respuesta medido (controlador Hill Climbing)

- Utilidad de los cambios realizados

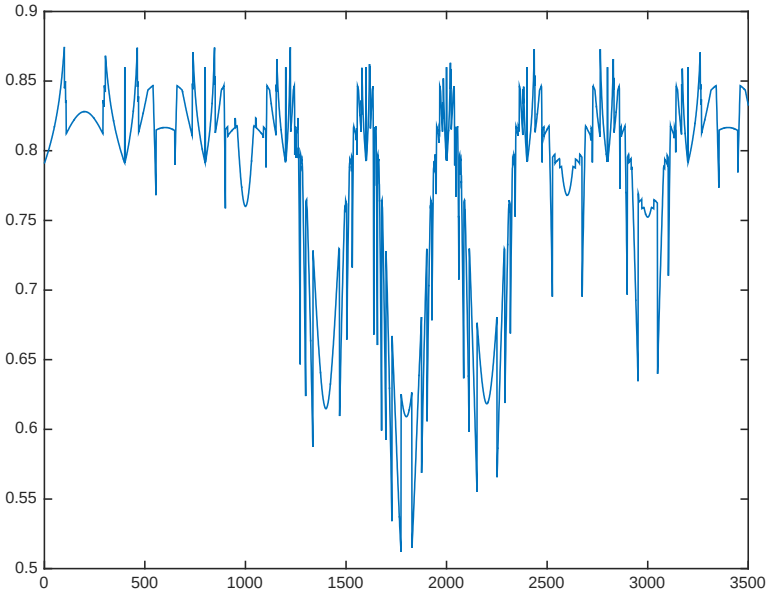


Figura 4.3: Utilidad de los cambios realizados (controlador Hill Climbing)

Coste medio	18.4851
Varianza / Desviación estándar	124.1527 / $\pm$ 11.1424
Nº incumplimiento SLA	0
Between tUp and tDown	975
Nº cambios de configuración	402
Nº cambios por iteración	0.1149

Basándonos en la figura 4.1, podemos observar que ésta es muy similar a la figura 3.6 referente al anterior controlador, pero en este caso se presentan ciertos cambios que optiman la configuración y con ello el coste de este controlador manteniendo nulo el número de incumplimientos del SLA.
Como desventaja el controlador Hill Climbing presenta menos de la mitad de iteraciones entre el tUp y el tDOwn. Si pensamos en el funcionamiento de este controlador, podemos apuntar que estos tiempos se encontrarán por debajo del tDown, lo que creará un sistema con tiempos demasiado rápidos e incluso puede llegar a ser excesivo. Destacar que, aunque las gráficas 4.1 y 4.2 sean muy similares a las figuras 3.6 y 3.7, y sobre todo en rango de iteraciones entre 1500 y 2000, realmente existe una oscilación de subida y bajadas de instancias (ampliar la figura para su percepción) que hacen que la utilidad de los cambios sea baja entre estas iteraciones [figura 4.3] respecto a la figura 3.8. Esto es debido a, como se ha comentado antes, que este algoritmo escala instancias hasta intentar mantenerlas entre el tUp y el tDown, pero puede ocurrir que una

configuración se encuentre por encima de tUp, se escale y las siguientes opciones de configuración produzcan tiempos por debajo del tDown, produciendo un sistema excesivamente rápido, característica la cual nuestra función utilidad penaliza.

4.2 Agresivo - relajado

En las secciones anteriores hemos visto dos tipos de controladores. El primero carga-respuesta que se basa en estudiar de antemano el funcionamiento del sistema para varias configuraciones y luego utilizar este conocimiento para crear un controlador de elasticidad sencillo. El segundo controlador basado en Hill Climbing requiere un conocimiento preciso del funcionamiento del sistema para intentar obtener un controlador óptimo. Para mantener la filosofía de la construcción de controladores simples proponemos utilizar el controlador agresivo-relajado propuesto por Aitor Gónzalez de Mendívil [20].

Este controlador se basa en ser agresivo a la hora de aumentar el número de instancias y ser relajado a la hora de disminuirlas, de ahí su nombre. Esto se debe a que, al aumentar instancias, aumenta una instancia a cada componente en la configuración existente en ese momento cuando el tiempo de respuesta excede del tiempo up. Si, por el contrario, el tiempo no alcanza al tiempo down, se elimina una instancia de un componente al azar. Si no se da ninguna de estas condiciones, la configuración se mantendrá. Destacar que los componentes no pueden soportar más de 10 instancias, por lo que no se podrá aumentar instancias este caso; por el contrario, como mínimo deben tener una instancia cada componente y tampoco se podrán eliminar instancias de estos componentes.

El algoritmo resultante para este controlador sería:

```
if t̂_R > tUp then
        if no están instanciadas el número máximo de instancias then
                Sumar una instancia a todos los componentes
                if un componente supera 10 instancia then
                        no aumentar instancias
                end
        end

else if t̂_R < tDown then
        calcular de forma random un integer de 1 a 6
        initialize k ← 0
        while (componente random tiene solo 1 instancia) y (k no igual a 6)
                calcular de forma random un integer de 1 a 6
                k ← k+1;
        end
        if componente random tiene más de 1 instancia
                Quitar una instancia a ese componente random
        end
end
```

Como observamos en el algoritmo existe un proceso random, el cual influye en los cálculos y con ello se consiguen diferentes configuraciones para la misma carga de trabajo. Para que sea visible este proceso, en este caso presentamos duplicados los resultados:

- Configuración por iteración

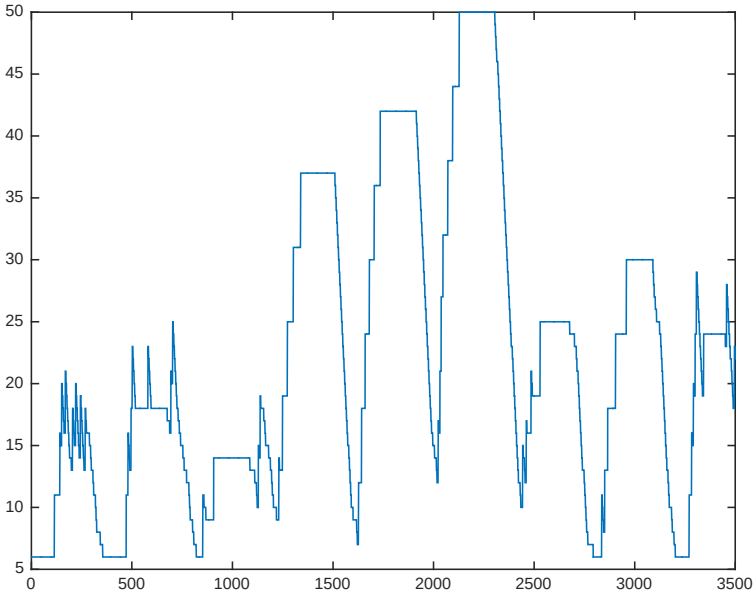


Figura 4.4.1: Gráfica del abastecimiento de instancias (controlador agresivo-relajado)

- Tiempo de respuesta

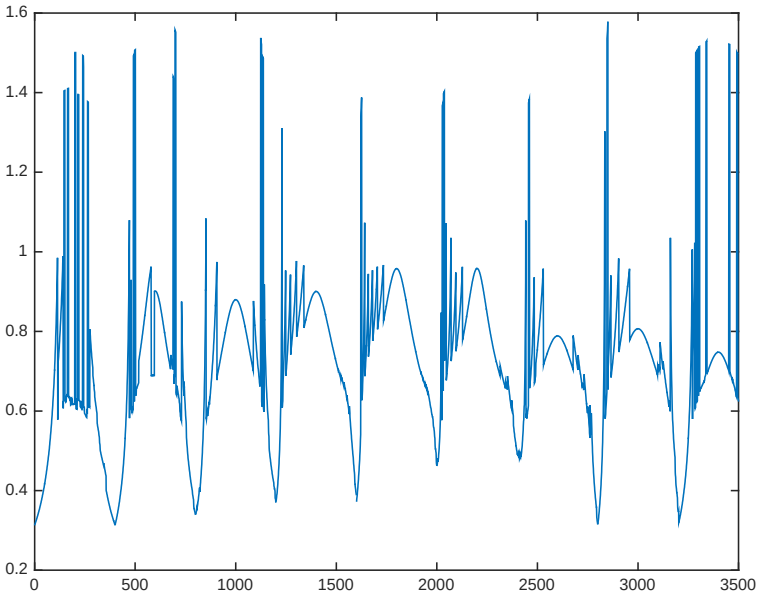


Figura 4.5.1: Tiempo de respuesta medido (controlador agresivo-relajado)

- Utilidad de los cambios realizados

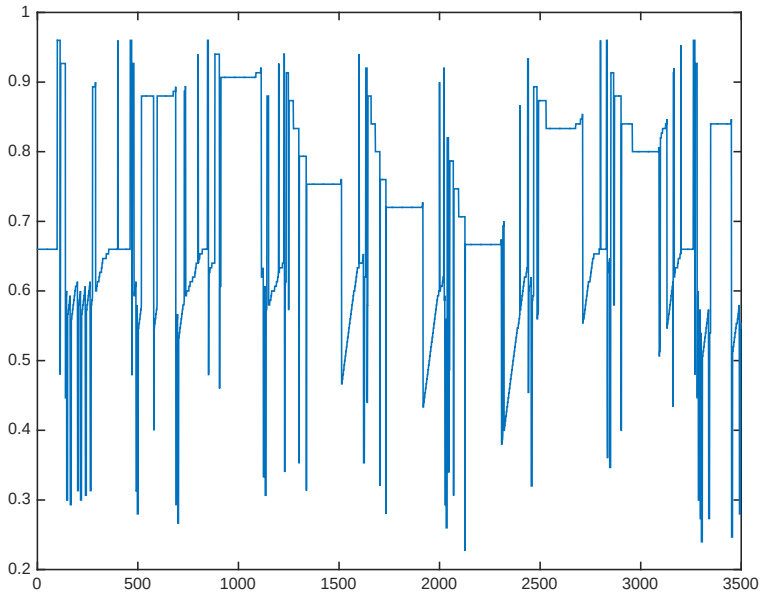


Figura 4.6.1: Utilidad de los cambios realizados (controlador agresivo-relajado)

Coste medio	22.0514
Varianza / Desviación estándar	152.9759 / $\pm$ 12.3683
Nº incumplimiento SLA	117
Between tUP and tDown	1927
Nº cambios de configuración	556
Nº cambios por iteración	0.1589

- Configuración por iteración

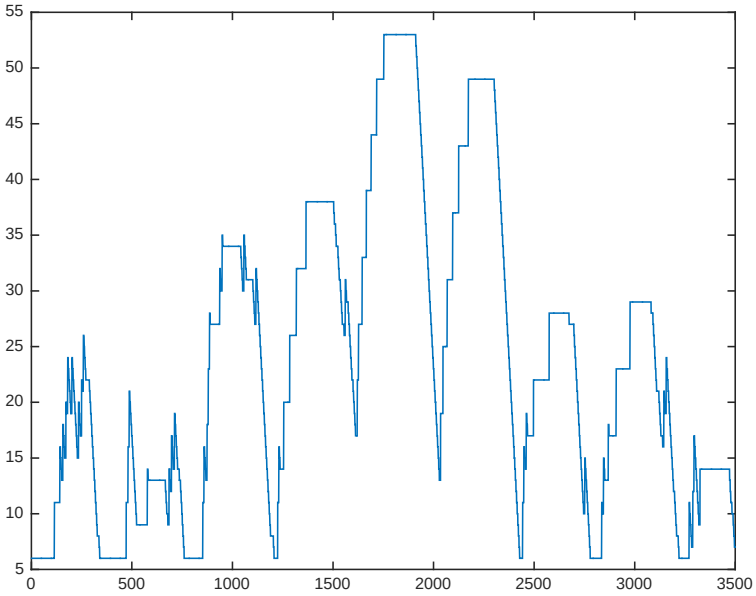


Figura 4.4.2: Gráfica del abastecimiento de instancias (controlador agresivo-relajado)

- Tiempos de respuesta

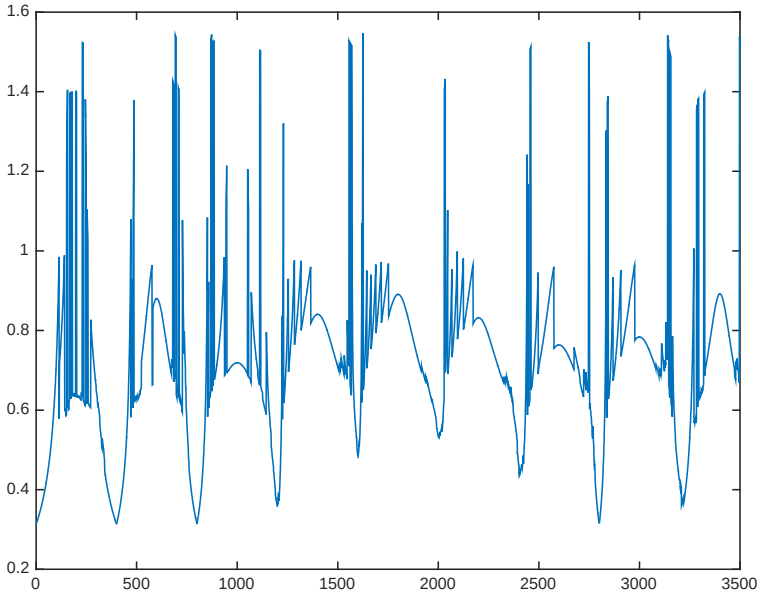


Figura 4.5.2: Tiempo de respuesta medido (controlador agresivo-relajado)

- Utilidad de los cambios realizados

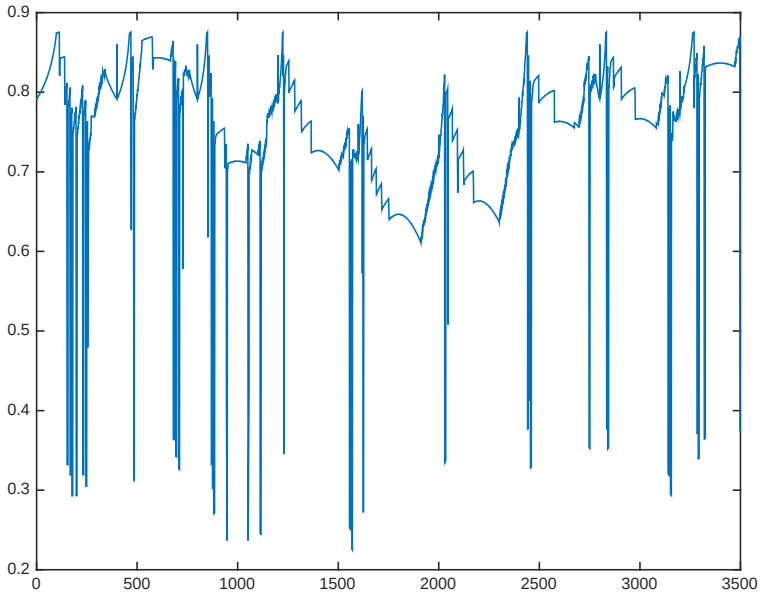


Figura 4.6.2: Utilidad de los cambios realizados (controlador agresivo-relajado)

Coste medio	23.4537
Varianza / Desviación estándar	189.1162 / $\pm$ 13.7519
Nº incumplimiento SLA	132
Between tUP and tDown	1921
Nº cambios de configuración	647
Nº cambios por iteración	0.1849

Observando las gráficas de utilidades, los cambios que realiza en la configuración este controlador determinan una utilidad mala, ya que varía mucho y con picos que llegan hasta una utilidad de 0.2. Estos picos son debidos a que el tiempo de respuesta se dispara y supera al tiempo SLA, lo cual, en nuestra fórmula de utilidad, cuanto más altos son los tiempos de respuesta más desfavorecidos resultan y, en consecuencia, utilidades más bajas.

Para obtener los resultados finales, hemos realizado un estudio estadístico en el que se han recogido los resultados 10 veces. Los resultados de este estudio estadístico son los que utilizamos para comparar entre los controladores lo cuales se plasman en la siguiente tabla:

Coste medio	23.5489
Varianza / Desviación estándar	162.4610/ $\pm$ 12,7254
Nº incumplimiento SLA	138,2 ≈ 138
Between tUP and tDown	1954,8 ≈ 1955
Nº cambios de configuración	623,3 ≈ 623
Nº cambios por iteración	0.1781

Observamos que en este caso aumenta el coste de instalar la configuración calculada por el controlador elástico a la vez que aumenta el número de cambios que realiza el sistema. Pero, aunque el número de iteraciones en las que el tiempo de respuesta se encuentra entre los tiempos up y down, aumenta a su vez el número de iteraciones en las que el tiempo de respuesta supera el SLA.

Como se ha comentado anteriormente, en este controlador existe un proceso random que influye en los cálculos, ya que no es lo mismo disminuir una instancia de un componente que tiene una demanda alta que eliminar una instancia que tiene una demanda baja. Por ello concluimos que los picos que se observan en la gráfica 4.4.1 y 4.4.2, son debidos a eliminar instancias de un componente con una importante demanda, lo cual perjudica en siguientes iteraciones, permitiendo no estar lo suficientemente abastecido el sistema.

4.3 Fuzzy

De acuerdo con el artículo “Self-Learning Cloud Controllers: Fuzzy Q-learning for Knowledge Evolution” [18], para el estudio de la elasticidad también se han propuesto controladores basado en conjuntos fuzzy junto con un algoritmo de aprendizaje. Nosotros nos hemos enfocado únicamente en la utilización de conjuntos fuzzy para la creación de controladores elásticos.

Los procesos fuzzy (difusos) consisten en asignar a un conjunto de entradas de control un conjunto de salidas de control a través de reglas difusas. El potencial de la lógica difusa reside en su capacidad para aproximar la no linealidad de un problema, expresando el conocimiento de manera similar a la percepción humana y el razonamiento [18]. Para nuestro controlador las entradas de control son la carga de trabajo u el tiempo de respuesta y la salida es la asignación de incremento o decremento en el número de instancias para los componentes.

El diseño de este controlador difuso implica las tareas:

1. Definir los conjuntos difusos y las funciones de pertenencia de las señales de entrada.
2. Definir la base de reglas que determina el comportamiento del controlador en términos de acciones de control utilizando variables lingüísticas definidas en la tarea anterior. El primer paso en el proceso de diseño es dividir el

espacio de estado de cada variable de entrada en varios conjuntos difusos mediante funciones de pertenencia, cada uno asociado a un término lingüístico como “bajo” o “alto”. La función de pertenencia $\mu(x)$ cuantifica el grado de pertenencia de una señal de entrada x al conjunto difuso y.

Los conjuntos para nuestro controlador quedan definidos así:

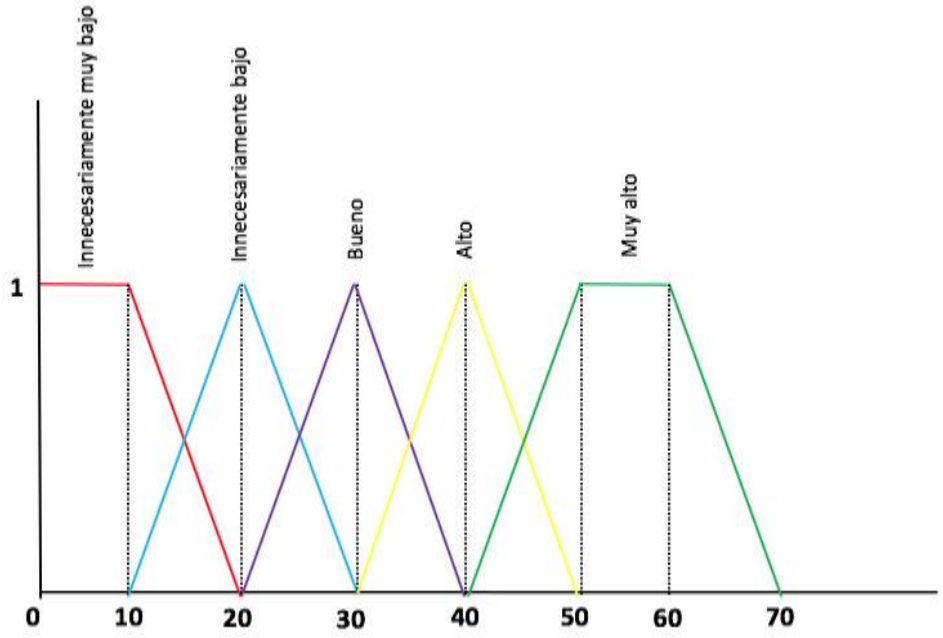


Figura 4.7: Conjunto fuzzy para el tiempo de respuesta.

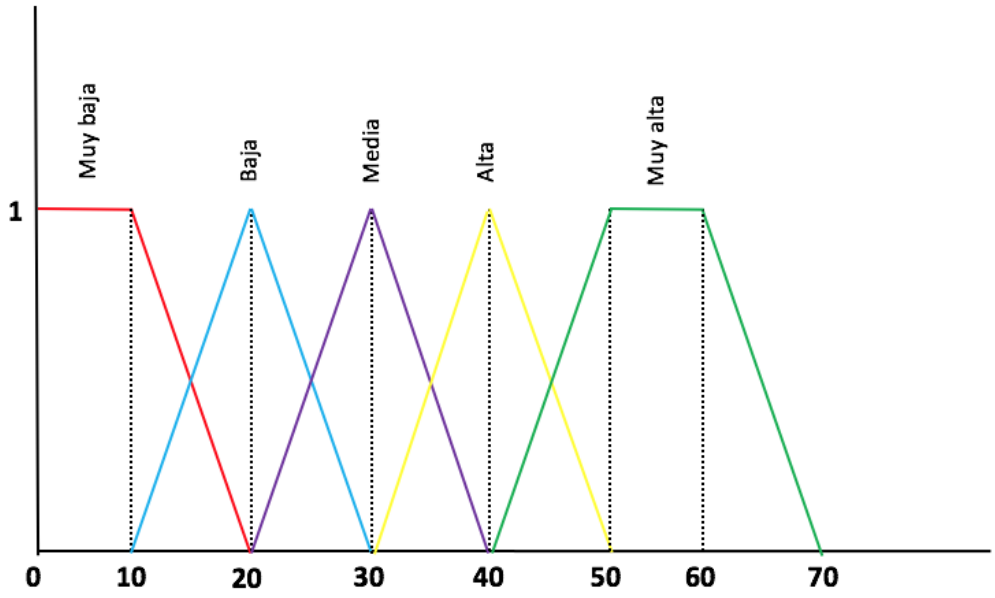


Figura 4.8: Conjunto fuzzy para la carga de trabajo.

Antes de calcular la pertenencia de un valor tiempo de respuesta o de carga, se halla el índice que tendría esos valores mediante una regla de tres, en la que el índice correspondiente a la carga máxima y al tiempo SLA será 60. Una vez hallados los índices correspondientes, se calcula la función de pertenencia $\mu(x)$.

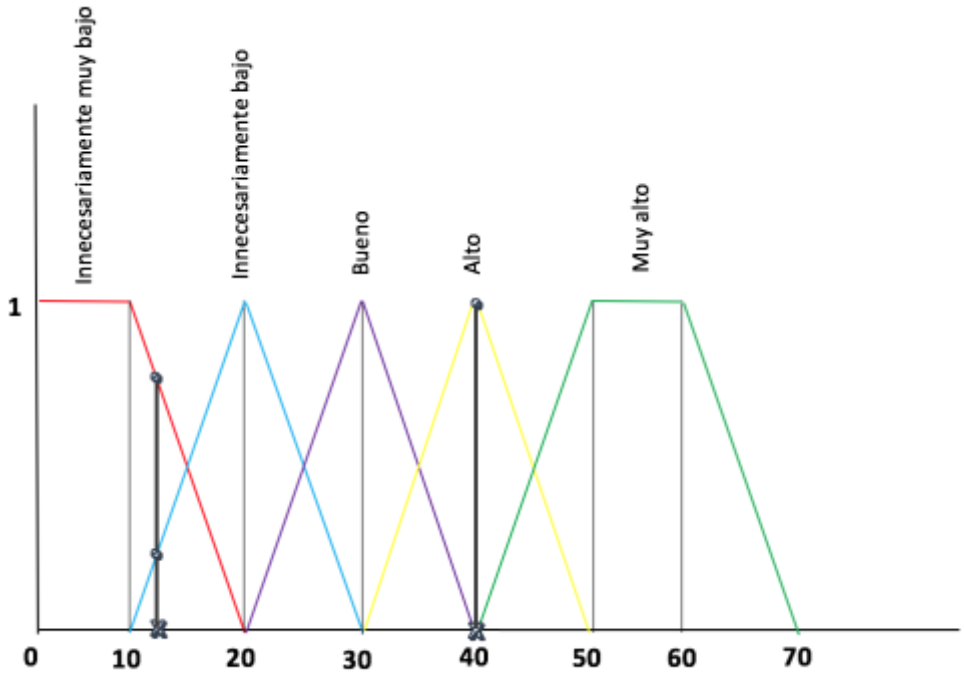


Figura 4.9: Representación gráfica de pertenencia de una señal X a un conjunto difuso Y

Para implementar la función $\mu(x)$, creamos una matriz de tamaño el número de reglas, en nuestro caso 5x5, que almacena el producto de los valores conjunto a conjunto. El diseño de esta implementación fue ideado para mantener constancia de la fuerza de las reglas.

$$\mu(x) = \begin{bmatrix} x_{rt1}^{w1} & x_{rt2}^{w1} & x_{rt3}^{w1} & x_{rt4}^{w1} & x_{rt5}^{w1} \\ x_{rt1}^{w2} & x_{rt2}^{w2} & x_{rt3}^{w2} & x_{rt4}^{w2} & x_{rt5}^{w2} \\ x_{rt1}^{w3} & x_{rt2}^{w3} & x_{rt3}^{w3} & x_{rt4}^{w3} & x_{rt5}^{w3} \\ x_{rt1}^{w4} & x_{rt2}^{w4} & x_{rt3}^{w4} & x_{rt4}^{w4} & x_{rt5}^{w4} \\ x_{rt1}^{w5} & x_{rt1}^{w5} & x_{rt1}^{w5} & x_{rt1}^{w5} & x_{rt1}^{w5} \end{bmatrix}$$

De tal forma que, para el ejemplo señalado, el valor de x_{rt4}^{w3} es el producto del grado de pertenencia de la carga de entrada con el conjunto difuso 3 para la carga y el grado de pertenencia del tiempo de respuesta con el conjunto difuso 4 para el tiempo de respuesta; o lo que es lo mismo, x_{rt4}^{w3} será el valor de la fuerza para la regla "carga media y tiempo de respuesta alto". Como se puede observar en la figura 4.9, un mismo valor puede tener dependencia a dos conjuntos, por lo que en esta matriz pueden estar

reflejadas hasta 4 reglas posibles (un valor de carga dependiente de dos conjuntos y un valor de tiempo de respuesta dependiente de dos conjuntos).

Acorde con los valores de la fuerza aparece una matriz en que se especifican las acciones que se deben realizar para las 25 posibles reglas. Nosotros hemos tomado para esta matriz los siguientes valores:

$$a_i = \begin{bmatrix} -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \\ -2 & -1 & 0 & 1 & 2 \end{bmatrix}$$

Como se puede observar, los valores de esta matriz se han determinado valorando únicamente las decisiones según el tiempo de respuesta, rebajando 2 instancias en el caso que tiempo de respuesta fuera "innecesariamente muy bajo" y añadiendo 2 instancias en el caso de que el tiempo de respuesta sea "muy alto". Así la acción a realizar para la regla ejemplo anterior sería añadir una instancia. Al poder existir hasta 4 reglas posibles, se necesita calcular cual es la acción correcta mediante la fórmula:

$$y(x) = \sum_{i=1}^{N} \mu_i(x) \times a_i$$

Donde N es el número de reglas, en nuestro caso 25.
Mediante este proceso obtendríamos la acción a realizar por el controlador fuzzy, el cual añadiría, eliminaría o mantendría igual la configuración teniendo en cuanta el mínimo y el máximo de instancias que permite el sistema.

El algoritmo de este controlador fuzzy descrito es:

Initialize a_i ← valores de las acciones a relizar por el controlador según una dependencia u otra

```
for i=1 to nº conjuntos para la carga
        almacenar en fila i de vector pertenencia de λ̂ con el conjunto i
end
for i=1 to nº conjuntos para el tiempo de repuesta
    almacenar en la fila i de vector pertenencia de t̂_R con el conjunto i
end

initialize f ← filas depWorkload;
initialize c ← filas depResponseTime;

for i=1: 1: número de conjuntos para el tiempo
    for j=1: 1: número de conjuntos para la carga
        dependencias(i,j) ← depWorkload(i) * depResponseTime(j);
    end
end
actionController ← suma de los valores obtenidos de dependencias x matriz acciones

if decision controlador es aumentar instancias then
    for i= 1 to longitude de N
        if el componente i puede admitir más instancias then
            sumar la acción del controlador al componente i
            if componente i tiene más del máximo de instancias then
                componente I con número máximo de instancias
            end
        end
    end
end

if decision controlador es eliminar instancias then
    for i=1 to longitude de N
        if componente i tiene más de una instancia y aplicando al componente la acción
          del controlador tiene al menos una instancia then
            sumar la acción del controlador al componente i
        else
            poner una instancia en componente i
        end
    end
end

if decision controlador mantener
    Nnew ← Nant;
end
```

Finalmente, los resultados obtenidos al aplicar el controlador fuzzy explicado en nuestro estudio son:

- Configuración por iteración

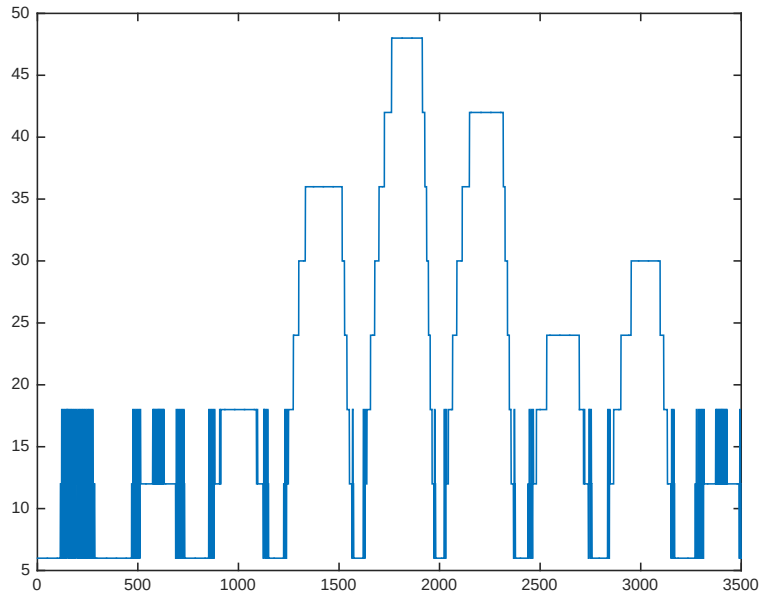


Figura 4.10: Gráfica del abastecimiento de instancias (controlador fuzzy)

- Tiempo de respuesta

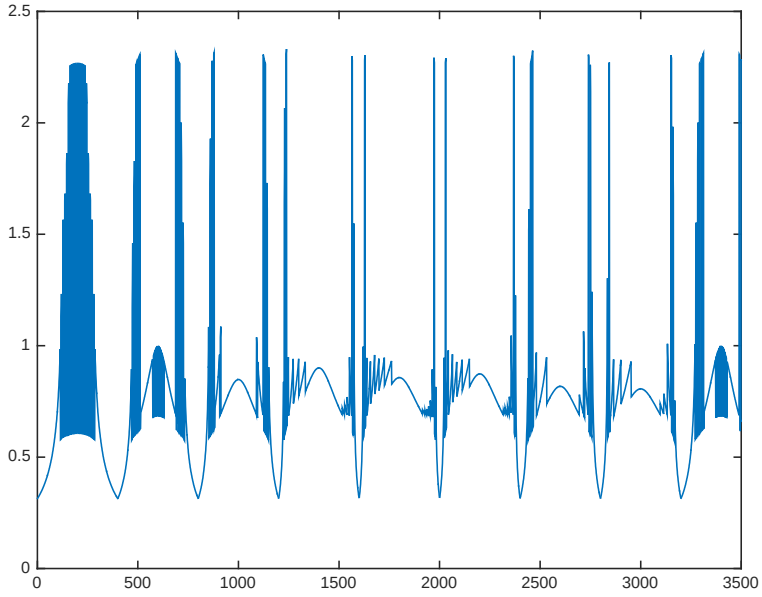


Figura 4.11: Tiempo de respuesta medido (controlador fuzzy)

- Utilidad de los cambios realizados

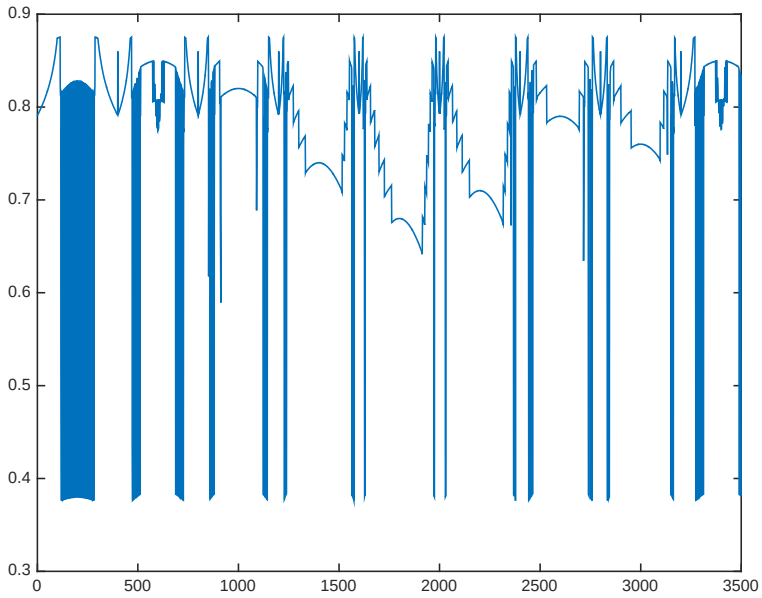


Figura 4.12: Utilidad de los cambios realizados (controlador fuzzy)

Coste medio	19.0889
Varianza / Desviación estándar	159.7395 / $\pm$ 12.6388
Nº incumplimiento SLA	170
Between tUP and tDown	1998
Nº cambios de configuración	2016
Nº cambios por iteración	0.5760

En el caso de este controlador, a diferencia del resto de controladores vistos, este consigue mantener el tiempo de respuesta entre los tiempos up y down un número mayor de veces, aunque proporcionalmente viola el SLA. Esto último puede ser posible ya que únicamente le permitimos al sistema aumentar 2 instancias por componente, y en ocasiones es necesario aumenta en una sola iteración más instancias. A pesar de esto, se obtienen resultados aceptables ya que el realiza aproximadamente una media de 0.58 cambios por iteración con un coste muy aceptable respecto a otros controladores.

Destacar que en las gráficas se observa que en un rango de carga baja (entre cargas de 0 a 10) este controlador oscila añadiendo y eliminando instancias continuamente. Esto se debe a que este controlador sólo es capaz de añadir y disminuir el mismo número de instancias por componente, y si en una iteración es necesario añadir en la siguiente iteración, aunque la carga de entrada aumente, es una configuración que sobre-abastece al sistema y por tanto determina en que se debe eliminar instancias en la

siguiente iteración. De esta oscilación se obtienen las oscilaciones en los tiempos de respuesta y en la utilidad de los cambios.

Capítulo 5

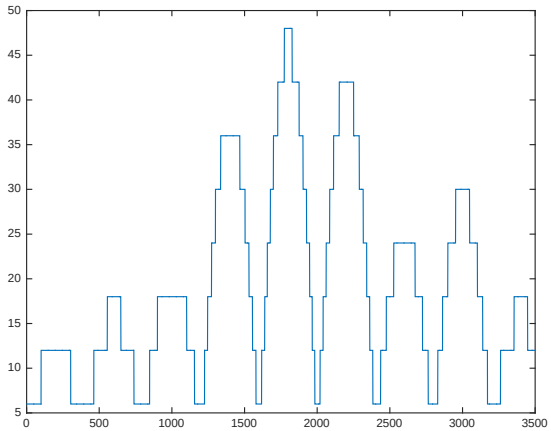
CONCLUSIONES.

En este trabajo hemos realizado un estudio sobre el comportamiento de controladores elásticos simples aplicados a un sistema de rendimiento, del que únicamente se sabe que es un sistema abierto multiclase, además de los resultados de tiempo que éste calcula con una configuración específica. Sobre este sistema se han aplicado diferentes controladores simples, todos ellos proactivos, y se han especificado los parámetros del tiempo de SLA, el cual no es aceptable que el sistema supere, y los tiempos Down y Up, que dictan que tiempos son muy favorables para el sistema. El primer controlador que aparece basado en la división de la carga de trabajo en porciones tras el estudio de carga-respuesta, el segundo basado en el algoritmo computacional de Hill Climbing, un tercero que aumenta un número muy alto de instancias y disminuye de una en una de forma aleatoria y un último controlador basado en conjuntos y reglas fuzzy. De todos ellos se han proporcionado las métricas que se han visto necesarias para una buena comparativa, como son el coste, la varianza y la desviación estándar, el número de incumplimientos del SLA, el número de iteraciones en las que el tiempo se encuentra entre el tiempo Down y el tiempo Up, el número de cambios que se realizan en la configuración y a su vez, el número de cambios que se producen por iteración. Visualmente presentamos 3 gráficas: la gráfica del abastecimiento de instancias, en la que se ve las diferentes configuraciones del sistema en cada iteración, la gráfica del tiempo de respuesta que calcula el sistema según la configuración en cada iteración y la gráfica de utilidad, que realiza el seguimiento de cómo tan buenos son los cambios de configuración en cada iteración.

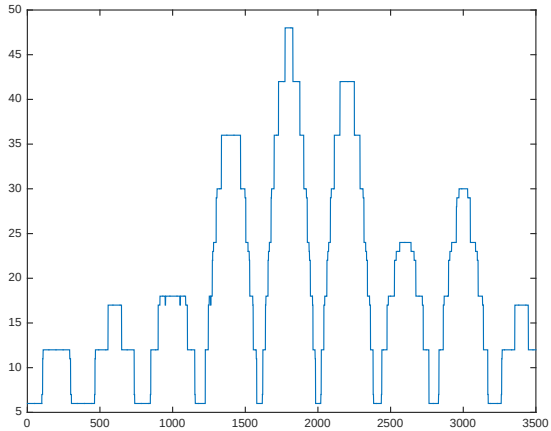
A continuación, recopilamos toda la información que los controladores nos han otorgado a lo largo del proyecto:

CONFIGURACIÓN POR ITERACIÓN

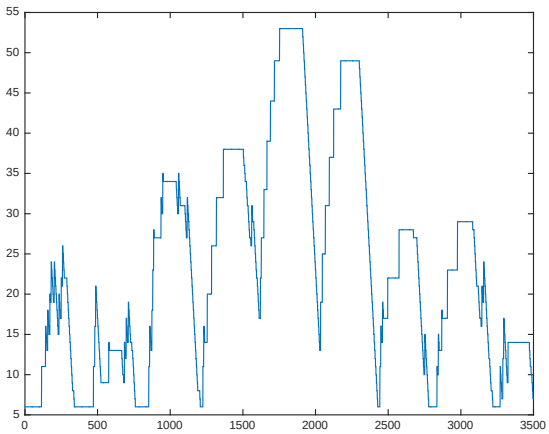
Controlador carga estimada



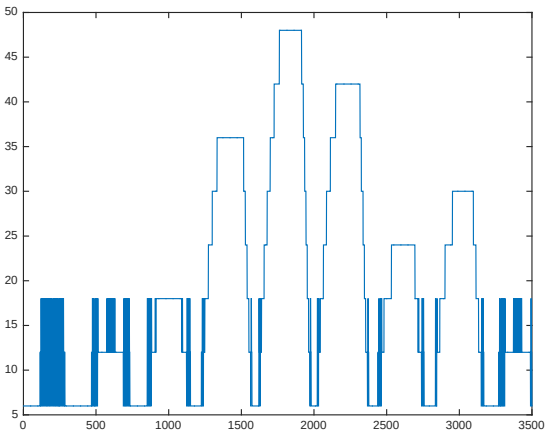
Controlador Hill Climbing



Controlador agresivo-relajado



Controlador fuzzy



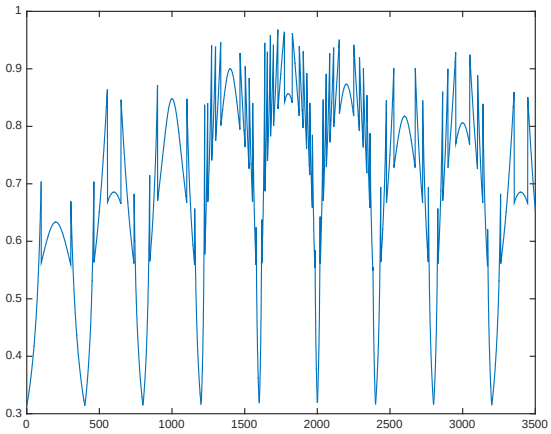
Si comparamos todos los controladores observando sus gráficas de configuración por iteración correspondientes a cada uno de ellos, podemos observar que las gráficas correspondientes a carga estimada y Hill Climbing parecen similares, ya que ambas mantienen la misma simetría y alcanzan el mismo número de instancias en las mismas iteraciones, permitiéndonos asegurar que la segunda afina la primera en ciertas iteraciones mejorando el comportamiento.

Tomando como bueno el patrón de éstas para una comparación con las otras dos, observamos que el controlador agresivo-relajado funciona correctamente ya que aumenta y disminuye cuando es preciso, pero cuando es necesario, es tan grande el aumento de instancias que sobre abastece el sistema, manteniendo este problema durante un notable número de iteraciones debido a que es muy lenta la disminución de instancias, y, en consecuencia, ir degradando el funcionamiento del controlador.

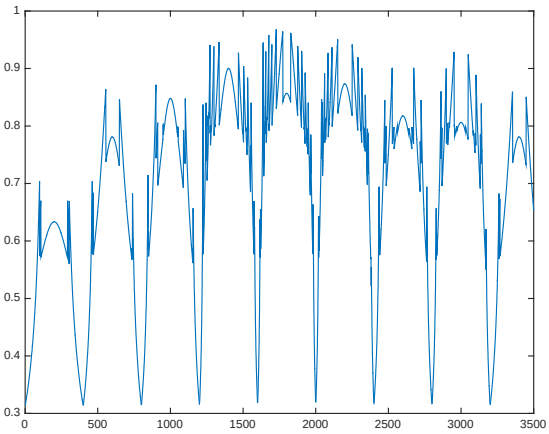
Por último, el controlador fuzzy aumenta y disminuye correctamente el número de instancias, pero se observa que en un cierto intervalo de carga de trabajo el controlador oscila subiendo y bajando el número de instancias, siéndole imposible un perfecto funcionamiento en comparación con los controladores carga estimada y Hill Climbing. Además de esta oscilación, se observa que este controlador no disminuye el número de instancias tan finamente como lo realizan los primeros. Dando respuesta alguna a este problema, a este controlador le cuesta un número importante de iteraciones tomar la decisión de disminuir el número de instancias.

TIEMPO DE RESPUESTA

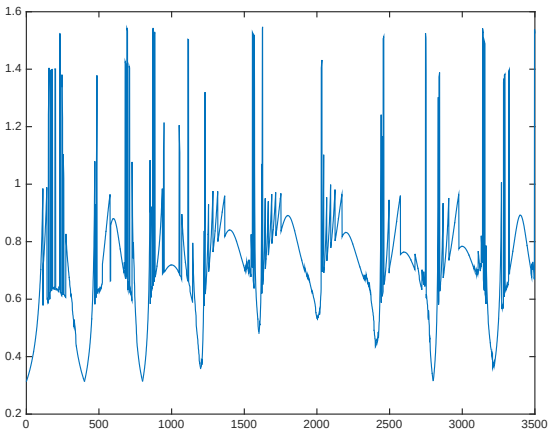
Controlador carga estimada



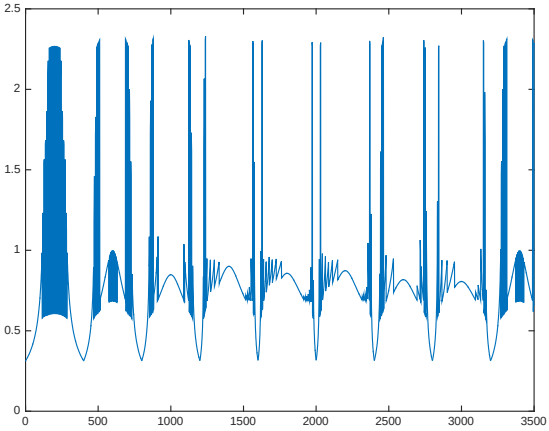
Controlador Hill Climbing



Controlador agresivo-relajado



Controlador fuzzy



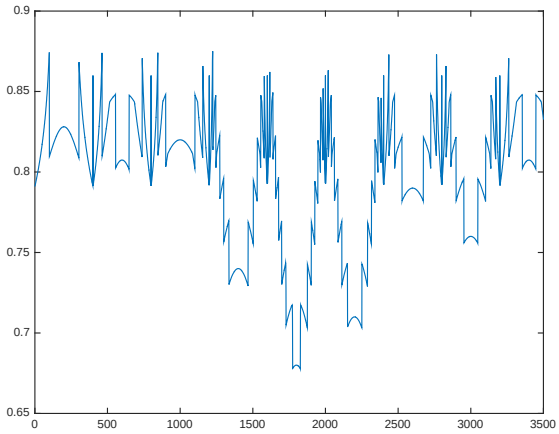
Las gráficas sobre la evolución del tiempo de respuesta de los controladores de carga estimada y de Hill Climbing, al igual que para las gráficas anteriores, son similares, manteniendo la simetría de los tiempos y tomando el Hill Climbing como mejora del controlador carga estimada. Se escogen estos patrones como buenos para poder comparar el resto.

Para el controlador agresivo-relajado, a pesar del problema sobre el abastecimiento de instancias que hemos visto antes, este problema no parece que le afecte en el tiempo de respuesta tanto como se podía pensar. Sí que en algunas iteraciones el tiempo de respuesta se dispara hasta llegar a un tiempo de respuesta de 1.5, sobre todo en las 1000 primeras iteraciones, pero parece ir estabilizándose conforme avanza, disparándose un número menor de veces.

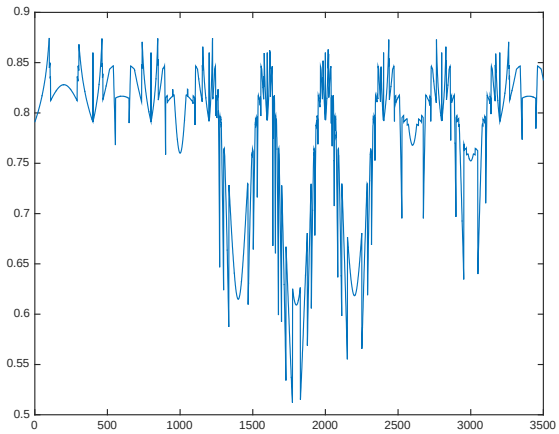
Por el contrario, al controlador fuzzy si le afecta el problema de oscilar subiendo y bajando el número de instancias en el tiempo de respuesta. En las iteraciones pertenecientes a las oscilaciones el tiempo se dispara gravemente alcanzando los 2.25 un número notable de veces.

UTILIDAD DE LOS CAMBIOS REALIZADOS

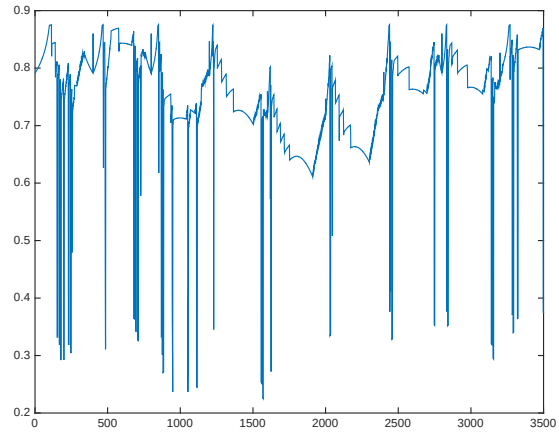
Controlador carga estimada



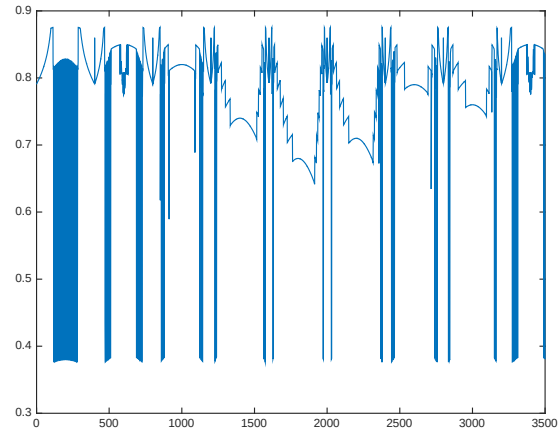
Controlador Hill Climbing



Controlador agresivo-relajado



Controlador fuzzy



Para el caso de las gráficas de la utilidad, las gráficas del controlador carga estimada respecto a la carga de Hill Climbing no se parecen tanto como en los casos anteriores, aunque si es cierto que mantienen una simetría parecida. Para la gráfica perteneciente al de carga estimada, todas las utilidades tienen un valor superior a 0.65 y para la gráfica perteneciente al Hill Climbing, las utilidades están por encima de 0.5. La respuesta a esto

es que el controlador Hill Climbing es más sensible al cambio de instancias y en ocasiones, no es tan útil afinar el número de instancias tanto, aunque debemos decir que mejora la gráfica de utilidad del controlador carga estimada, ya que, a pesar de lo comentado, la del Hill Climbing tiene durante más veces la utilidad entre el intervalo [0.8, 0.85].

Para el controlador agresivo-relajado, el ser agresivo a la hora de subir instancias afecta de forma importante en su utilidad, ya que le crean utilidades muy bajas, llegando a alcanzar utilidades por debajo de 0.3.

Por último, el controlador fuzzy oscila las utilidades a la vez que oscila en la subida y bajada de instancias, conteniendo la utilidad en el intervalo [3.7, 8.5].

TABLA COMPARATIVA DE MÉTRICAS

	CARGA ESTIMADA	HILL CLIMBING	AGRESIVO - RELAJADO	FUZZY
Coste medio	18,7217	18,4851	23,5489	19,0889
Varianza / desviación estándar	123,7968 / ± 11,1264	124,1527 / ± 11,1424	162,4610/ ± 12,7254	159,7395 / ± 12,6388
Nº incumplimiento SLA	0	0	138,2 ≈ 138	170
Between tUP and tDown	1867	975	1954,8 ≈ 1955	1998
Nº cambios de configuración	378	402	623,3 ≈ 623	2016
Nº cambios por iteración	0,1080	0,1149	0,1781	0,5760

Los resultados de la tabla plasman lo comentado hasta ahora, ya que, los dos primeros son muy parecidos, mejorando en algunos aspectos el Hill Climbing al controlador carga estimada.

El agresivo-relajado dispara su coste, pero mantiene durante un número de veces mayor el tiempo entre el intervalo [tDown, tUp] a la vez que aumenta el número de cambios, aunque viola 138 veces el tiempo SLA.

Para el controlador fuzzy, a pesar de su comportamiento oscilatorio, los resultados son aceptables, ya que la diferencia de su coste con los dos primero es pequeña; Además, es el controlador que más veces mantiene el tiempo entre los tiempos tDow y tUp, manteniéndolo 1998 veces, y el número de cambios que produce, siendo este 2016; aunque también es el controlador que más veces viola el tiempo SLA con un número de 170 veces.

Se puede decir que los resultados recogidos y que aportan los controladores son buenos, concluyendo que cualquier controlador elástico con las características desarrolladas en el trabajo es buena solución para aplicaciones en la nube. Ciertamente se debería tener en cuenta a qué se da más importancia, ya que por ejemplo el Hill Climbing a simple vista puede ser el mejor, pero en 3500 iteraciones ni la mitad de las veces se encuentra entre el tiempo Up y Down.

Cabe destacar que, en trabajos futuros, se podrían someter a estos controladores a técnicas de aprendizaje, como puede ser el Qlearning, y con ello obtener mejores valores, o, aunque sea complejo, someterlos a técnicas que no sean ideales para ver su funcionamiento de una forma más real.

FIGURAS

BIBLIOGRAFÍA

[1] Computación en la Nube, https://es.wikipedia.org/wiki/Computación_en_la_nube.

[2] Amazon AWS: https://aws.amazon.com/es/?nc2=h_lg

[3] Openstack: https://www.openstack.org/

[4] Google Compute Engine: https://cloud.google.com/?hl=es

[5] Google App Engine: https://cloud.google.com/?hl=es

[6] Heroku: https://www.heroku.com

[7] Azure: https://azure.microsoft.com

[8] Bitnami: https://bitnami.com/

[9] Salesforce: https://www.salesforce.com

[10] Atlassian Cloud: https://www.atlassian.com

[11] N. Herbst, et. Al. Elasticity in Cloud Computing. What It is, and What It is not. 10th International Conference on Autonomic Computing (ICAC 13). pp. 23-27, 2013.

[12] Daniel A. Menascé, Virgilio A. F. Almeida y Lawrence W. Dowdy. "Performance by Design. Computer Capacity Planning by Example", PRETINCE HALL PTR, January 2004.

[13] B.Schneiderman, "Response time and display rate human performance with computers", Computing Surveys, ACM, vol. 16, no. 3, September 1984.

[14] R. B. Miller, "Response time in man-computer conversational transactions", Proc. AFIPS Fall Joint Comp. Conf.,33:267-277, December 1968.

[15] J. Nielsen, Usability Engineering, Morgan Kaugmann, San Francisco, California, 1994.

[16] Mahmoud Awad, Daniel A. Menascé, "Dynamic Derivation of Analytical Performance Model in Autonomic Computing Enviroments", Volgenau School of Engineering, George Mason University, USA, 2014

[17] Algoritmo Hill Climbing, https://es.wikipedia.org/wiki/Algoritmo_hill_climbing

[18] Pooyan Jamshidi, Amir Sharifloo, Claus Pahl, Andreas Metzger, Giovani Estrada, "Self-Learning Cloud Controllers: Fuzzy Q-learning for Knowledge Evolution", Imperial

College London,UK, Dublin City University, Ireland, University of Duisburg-Essen, Germany, Intel,Ireland, 2 July 2015.

[19] Yasir Shoaib and Olivia Das. Performance-oriented Cloud Provisioning: Taxonomy and Survey arXiv:1411.5077v1 [cs.DC] 19 Nov 2014

[20] Aitor González de mendívil, “Controlador de Elasticidad Basado en Técnicas de Clasificación Binaria”, 18 junio de 2017.