
UNIVERSIDAD DE VALLADOLID

ESCUELA TÉCNICA SUPERIOR

INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

MASTER EN INGENIERÍA DE TELECOMUNICACIÓN

# Implantación y evaluación de un entorno de simulación de redes distribuido basado en ns-3 y computación en nube

Autor:

**D. Sergio Serrano Iglesias**

Tutor:

**Dr. D. Eduardo Gómez Sánchez**
**Dr. D. Miguel Luis Bote Lorenzo**

Valladolid, 21 de Septiembre de 2017

| | |
|---|---|
| TÍTULO: | **Implantación y evaluación de un entorno de simulación de redes distribuido basado en ns-3 y computación en nube** |
| AUTOR: | **D. Sergio Serrano Iglesias** |
| TUTOR: | **Dr. D. Eduardo Gómez Sánchez**<br>**Dr. D. Miguel Luis Bote Lorenzo** |
| DEPARTAMENTO: | **Teoría de la Señal y Comunicaciones e Ingeniería Telemática** |

**Tribunal**

| | |
|---|---|
| PRESIDENTE: | **Dr. D. Juan Ignacio Asensio Pérez** |
| VOCAL: | **Dr. D. Manuel Rodríguez Cayetano** |
| SECRETARIO: | **Dr. D.ª Luisa María Regueras Santos** |
| FECHA: | **21 de Septiembre de 2017** |
| CALIFICACIÓN: | |


## Resumen del TFM

Las simulaciones de barrido de parámetros ofrecen un gran potencial en el estudio de las redes de comunicación, tanto en investigación como en el nivel educativo, pero requieren de un elevado tiempo de ejecución por la gran cantidad de simulaciones individuales que resultan de la explosión combinatoria de dar diversos valores a algunos de sus parámetros. En este TFM se ha trabajado en el DNSE3, un entorno de simulación que hace uso de la nube computacional para procesar de forma distribuida las simulaciones para reducir sus tiempos de ejecución. Su diseño, tanto de la arquitectura como del funcionamiento interno, se ha preparado para integrarse dentro de diferentes sistemas de nube y para explotar su escalado, ofreciendo un mecanismo de asignación de trabajos independiente del número de recursos desplegados y también una serie de reglas que ofrezcan un auto-escalado eficiente, aprovisionando y eliminando las instancias según sea necesario. Con el sistema final operativo, se ha constrastado su rendimiento y usabilidad a traves de pruebas sintéticas y con usuarios reales, respectivamente, con los ofrecidos con el uso local del simulador.



## Palabras clave

computación distribuida, nube computacional, simulación de redes de ordenadores, escalabilidad automática.



## Abstract

Parameter swepp simulations have a great potential in understanding computer networks, in both research and education, but they convey a significant computation load due to the great amount of individual tasks resulting form the different combination of parameters. This Master Thesis proposes DNSE3, a simulation environment that makes use of cloud computing to exploit parallelization in order to achive reduce response times for parameter sweeps. Its design, both architecture and internal behavior, has been prepared to integrate through differente cloud infrastructres, featuring a decentralized scheduling mecanism that, along with aplication level workload metrics and user defined scalability rules, achieves self-scalability, by provisioning and freeing resources for simulations as needed. With the final operating version of the system, performance and usability have been compared with those offered with the local use of the simulator though synthetics tests and real users, respectively.

## Keywords

distributed computacion, cloud computing, computer networks simulation, automatic scalability

# Agradecimientos

A cada año que pasa, se incrementa el número de personas a las que tengo que agradecer el haber llegado hasta aquí. En primer lugar, gracias a Miguel y a Eduardo, que continúan aguantando mi particular estilo de trabajo y apoyándome en los diferentes bloqueos.

También quiero agradecer a Juan Ignacio y a Manuel, por acompañarme en este proyecto y no tirarme demasiado de las orejas por los diferentes "problemas" que iban apareciendo.

Gracias de corazón a todo el grupo de investigación GSIC/EMIC, por amenizar las largas tardes en las que me quedaba lanzando simulaciones una y otra vez. Nunca dejarán de sorprenderme la actitud y energía que irradian todos y cada uno de sus miembros.

También quiero agradecer a todos mis compañeros de penurias y alegrías a lo largo de la carrera y el máster. Agradezco el ánimo y apoyo de los que han continuado aquí en Valladolid como los que están arrasando allá donde van y que se les extraña todos los días.

No puedo olvidarme de Adrián, Adrián, Diego, Victor y Victor. Pasarán los años y seguiré pudiendo contar con vosotros para pasar buenos ratos y compartir grandes experiencias, aunque nos veamos de poco en poco.

Por último, pero no menos importante, quiero agradecer la ayuda, el cuidado y la preocupación de toda mi familia. Pasarán los años, pero nunca agradeceré lo suficiente todo el apoyo y el amor que me han brindado, tanto en los buenos como en los malos tiempos.

# Índice general

# Índice de figuras

# Índice de tablas

# Capítulo 1

# Introducción

## 1.1. Motivación

Una de las herramientas más empleadas en el estudio de las redes de ordenadores son las simulaciones. Empleadas tanto por investigadores y empresas, posibilitan el estudio de los protocolos y de las configuraciones red sin necesidad de disponer de una infraestructura real que represente el entorno objeto de estudio, que puede ser demasiado costosa o compleja de desplegar [Law91, Wei09], o bien revisar situaciones de tráfico anómalas y poco frecuentes que podrían afectar de forma significativa al correcto funcionamiento del resto de nodos presentes en la red. Una muestra de su relevancia dentro del campo es la gran variedad de simuladores disponibles, tanto de código abierto como de licencia propietaria, como son el ns-2[1], ampliamente usado en la investigación, su sucesor el ns-3[2], OMNet++[3] o Riverbed Modeler[4].

Su utilización no se ve restringida exclusivamente a la industria y a la investigación; los entornos académicos se aprovechan de éstas con fines pedagógicos para afianzar los conocimientos de los alumnos. Estos pueden, por ejemplo, replicar los escenarios planteados durante las sesiones teóricas en diversos modelos con los que configurar libremente las condiciones del estudio. Estos modelos ofrecen una mayor flexibilidad que los bancos de pruebas, formados por una serie de equipos previamente preparados con los que generar diferentes comunicaciones entre ellos y con las redes externas. Estos bancos no solo incurren en los problemas de costes mencionados anteriormente, sino que limitan los escenarios que pueden revisar los alumnos, no se pueden utilizar fuera de las sesiones de laboratorio y requieren de medidas cautelares de seguridad para que las puedan utilizar. Así, los simuladores han sido usados, por ejemplo, para mostrar como hace [Pap02] el uso del ns-2 de forma conjunta en las sesiones de teoría, donde expone algunos de los conceptos impartidos, y en las sesiones prácticas, donde los alumnos evalúan el impacto de los diferentes protocolos. En el caso de [Qun08] se añade una interfaz gráfica al ns-2 que facilite a los alumnos su uso en la revisión de los resultados. De forma similar, [Wan12] hace uso del ns-3 para ilustrar a los alumnos del comportamiento dinámico y las métricas

[1] `http://www.isi.edu/nsnam/ns/`

[2] `https://www.nsnam.org/`

[3] `https://omnetpp.org/`

[4] `https://www.riverbed.com/es/products/steelcentral/steelcentral-riverbed-modeler.html`

de rendimiento de las redes inalámbricas locales.

En todos los ejemplos mencionados hasta ahora se propone un reducido conjunto de configuraciones que pueden simular los alumnos, cuando la exploración de los parámetros de los modelos para detectar fenómentos relevantes y comprender sus casuas puede enriquecer su aprendizaje [BL12]. Este tipo de simulaciones son las llamadas simulaciones de barrido de parámetros, en las que uno o más de sus parámetros de entrada toma valores dentro de un rango definido en la ejecución para dar muestra del efecto producido en los resultados generados.

El problema que presentan esta clase de simulaciones es el gran incremento en el tiempo de respuesta hasta que se dispone de los resultados finales [BL12]. La ejecución de estas simulaciones implica la realización de múltiples simulaciones individuales, tantas como número de combinaciones diferentes presentan los valores de los parámetros, de tal forma que, aunque cada una de ellas se complete en un corto periodo de tiempo, este volumen retrasa la obtención de los resultados de todas ellas. Si este tiempo es excesivamente largo puede restringir su uso en las sesiones de laboratorio, con tiempo muy limitado que no puede desaprovecharse haciendo esperar a los alumnos a su finalización.

Es interesante reparar en el hecho de que cada una de las simulaciones originadas en estos barridos constituye un proceso único que es totalmente independiente del resto del conjunto. Aunque todas ellas hacen uso del mismo modelo, no presentan ninguna dependencia con las salidas originadas tras la ejecución de otras, lo que abre la puerta a soluciones *hardware* que aprovechen la paralelización de alto nivel presente en todos estos procesos [Wan12]. Sin embargo, el límite de paralelización que logra alcanzar esta solución se ve rápidamente alcanzado. Incluso si se decide mejorar los recursos disponibles en la propia máquina, con los costes y complicaciones que acarrea este cambio de *hardware*, sigue siendo inviable la ejecución de trabajos excesivamente largos.

En lugar de recurrir a la paralelización *hardware* dentro de una sola máquina se pueden migrar las simulaciones a un entorno distribuido donde se distribuya este trabajo entre los diferentes nodos que lo conforman. A diferencia de la paralelización *hardware*, estos sistemas pueden ser escalados con una menor complejidad al solo tener que agregar o eliminar máquinas del sistema, por lo que pueden adaptarse mejor a los picos cortos e intermitentes de demanda de trabajo. Sin embargo, su uso no está exento de complicaciones. Dado que cada uno de los nodos que componen el sistema trabajan de forma asíncrona e independiente del resto, se necesita incorporar un mecanismo que garantice el reparto de los trabajos entre todos ellos. Por otro lado, no todas las tareas alcanzan el mismo rendimiento en su ejecución distribuida. Aquellas tareas que, en su fragmentación en subprocesos, muestren una dependencia con la salida generada por alguno de los subprocesos anteriores necesitarán de un sistema a mayores que garantice la sincronía entre los nodos en su ejecución. Por suerte, las tareas que atañen a este trabajo, es decir, las simulaciones de barrido de parámetros no presentan este problema por la independencia de las simulaciones originadas, como se mencionó anteriormente.

Ante esta necesidad de reducir los tiempos de respuesta de los barridos de parámetros, [BL12] presentó el DNSE (*Distributed Network Simulation Environment*), un entorno de simulación orientado a la ejecución de simulaciones de barrido sobre el ns-2. Esta aplicación ofrecía una arquitectura orientada a servicios sobre una infraestructura de grid computacional [BL04, Fos01] de forma que se podía regular el número de servicios de

simulación desplegados en los diferentes nodos según fuese necesario. Este entorno fue utilizado dentro de los planes antiguos del Título de Ingeniero de Telecomunicación e Ingeniero Técnico de Telecomunicación de la Universidad de Valladolid con una gran acogida por parte de los instructores y sus alumnos. Sin embargo, presentaba una serie de problemas estrechamente relacionados con su infraestructura. Aunque el sistema facilita el escalado, se requiere la labor de un administrador para poner a punto los nuevos nodos, incurriendo a situaciones de sub y sobredimensionado mientras fluctua la carga de trabajos. Por otro lado, existía una considerable dificultad técnica y administrativa para la participación de varias organizaciones para compartir sus dispositivos, lo que restringió al uso de las máquinas en propiedad de la universidad para las pruebas de [BL12]. Finalmente, los *middlewares* disponibles para la administración y gestión del grid no tenían la suficiente madurez para el desarrollo eficiente de aplicaciones basadas en esta infraestructura.

Uno de los paradigmas de computación distribuida que ha captado más la atención en los últimos años es la nube computacional [Arm10, Buy09]. Recurriendo a las técnicas de virtualización, la nube es capaz de proveer a sus usuarios de una gran variedad de recursos computacionales, como instancias, dispositivos de almacenamiento, redes privadas y enrutadores; y abstrayéndolos de la gestión y administración de la infraestructura subyacente. Gran parte de la popularidad de la que goza se debe a la oferta de nubes públicas, como Amazon Web Services (AWS)[5], Microsoft Azure[6] o Google Compute Engine[7] entre otras; que ofrecen estos recursos bajo un modelo de coste por uso (*pay-per-use*) y al rápido aprovisionamiento bajo demanda de estos [Arm10, Vaq08]. Dada la gran acogida que ha recibido la nube, los *middlewares* de gestión de la nube han ido incorporando diferentes funciones demandadas por sus usuarios, como es el caso de los sistemas de escalado. Gracias a la definición de unas reglas que revisan diferentes métricas relacionadas tanto con el uso de la infraestructura como con la aplicación desplegada [Vaq13] se pueden desplegar sistemas capaces de redimensionar los recursos empleados de forma totalmente automática, a diferencia de lo que ocurría en el grid.

Incluso si se hace uso de una nube privada para estos despliegues, sus sistemas de gestión, como OpenStack[8] o CloudStack[9], ofrecen compatibilidad con nube públicas, como es el caso de AWS para los sistemas anteriormente mencionados. De esta forma, cabe la posibilidad de disponer de un sistema de nube híbrida donde la mayoría de los recursos lanzados se ejecutan dentro de la infraestructura privada y, cuando la demanda de trabajo exceda la capacidad soportada por la infraestructura real, se deleguen nuevas instancias en la nube pública, disfrutando de un escalado flexible y virtualmente infinito con unos costes razonables [Zha10].

Con esta nueva infraestructura, más popular, con capacidad de escalado automático y acceso a una cantidad virtualmente infinita de recursos y con *softwares* más robustos que en el grid, se inició el desarrollo del DNSE3 (*Distributed Network Simulation Environment 3*) [CP12, SI16], una continuación del DNSE donde ahora se utiliza la nube para la ejecución distribuida de los modelos del simulador ns-3. Al igual que su antecesor, es-

[5] https://aws.amazon.com/es/
[6] https://azure.microsoft.com/es-es/
[7] https://cloud.google.com/compute/?hl=es
[8] `https://www.openstack.org/`
[9] `https://cloudstack.apache.org/`

te sistema presenta una arquitectura orientada a servicios diseñada para sacar provecho de los recursos ofertados en la nube. En el transcurso del trabajo realizado en [SI16] se continuó con la propuesta de diseño presentada por [CP12], que solo ofrecía un diseño conceptual de la arquitectura de servicios diseñada en los albores de la nube con unos sistemas de gestión más básicos que los disponibles actualmente. Entre las metas logradas se incluían la revisión de la arquitectura del sistema que reutilizase, en la medida de lo posible, los servicios ofrecidos por los *middlewares*, quienes ofrecen soluciones propias ya adaptadas para la infraestructura que se utilizaba (como es el caso del almacenamiento persistente de los datos de la aplicación o el escalado de los recursos de simulación); la propuesta de un método de asignación de las tareas de simulación, que se basó en el mecanismo de *work-stealing* para simplificar las labores de escalado sin necesidad de mantener un registro de los recursos disponibles en todo instante de tiempo; y el despliegue de un prototipo funcional del sistema. Con esta implementación preliminar se pretendía, por un lado, comprobar el uso de un sistema de escalado basado en la publicación de métricas generadas por la propia aplicación y, por otro, demostrar la viabilidad de la arquitectura para reducir los tiempos de simulación percibidos en las simulaciones de barrido de parámetros, a pesar de que fueran algo comedidos.

Aunque este prototipo era completamente funcional e incluía las funciones esenciales del sistema, presentaba una serie de limitaciones que imposibilitaban su uso con usuarios reales. Aparte de la falta de persistencia de los datos de los usuarios no almacenados en el servicio de almacenamiento, la ausencia un cliente gráfico que simplificase el uso de la aplicación o la no inclusión de un sistema de autenticación y autorización de los usuarios, la versión presentada del DNSE3 presentaba ciertas deficiencias en la asignación de tareas y en las políticas de escalado. Cuando se disponía de varias instancias de simulación activas, se detectaron accesos simultáneos a una misma tarea por parte de varias instancias que supuso un detrimento en el rendimiento experimentado. En lo que respecta a la política de escalado, esta estaba destinada única y exclusivamente a la evaluación del mecanismo de escalado ofertado por la nube y no podía ser utilizada en un entorno real por el derroche de recursos que originaría, ya que generaba de forma continua nuevas instancias de simulación siempre que estuviera presente algún trabajo pendiente de completar.

En este Trabajo Fin de Máster (TFM) se abordan las limitaciones anteriormente mencionadas con el propósito de obtener una versión final apta para su uso en las asignaturas de “Ingeniería de Teletráfico en Redes Telemáticas” y “Teletráfico” impartidas en los Grados en Ingeniería de Tecnologías y Tecnologías Específicas de la Universida de Valladolid. Para ello, se revisa el mecanismo de asignación de trabajos para garantizar que solo sea accedido y procesado por un único servicio de simulación y se proponen unas nuevas reglas de escalado que garanticen el uso eficiente de los recursos de la nube. Después de superar estas limitaciones y disponer de un despliegue real del sistema, se evalua el comportamiento ofrecido frente a la solución actualmente disponible en las sesiones de laboratorio de las asignaturas anteriores. A diferencia de las pruebas de viabilidad realizadas en [SI16], en estas pruebas se recurre a cargas sintéticas de trabajo para medir su rendimiento y a usuarios reales para juzgar la usabilidad.

## 1.2. Objetivos

En la sección anterior se indicó la presencia de ciertas deficiencias en el prototipo del DNSE3 disponible al término de [SI16]. Los objetivos establecidos para este TFM están orientados a la superación de estas limitaciones y son:

- **La revisión de los mecanismos de planificación y escalado de las simulaciones**. La planificación de los trabajos de simulación garantizará la disposición de cada una de estos trabajos a un servicio de simulación diferente sin incurrir en colisiones de peticiones, siempre y cuando el número de trabajos lo permita. Por otro lado, el sistema de escalado deberá garantizar el uso responsable de los recursos de la nube y facilitar al administrador del sistema la libre configuración de sus reglas para acomodarlas al uso que recibirá el DNSE3.
- **El diseño y desarrollo de una interfaz de usuario** cómoda y agradable de usar por los estudiantes de las asignaturas de teletráfico. Esta interfaz facilitará el acceso de los usuarios al sistema, de tal forma que puedan lanzar y analizar las simulaciones de barrido de parámetros que generen.
- **Introducir las modificaciones oportunas en el desarrollo del DNSE3**. Estas modificaciones garantizarán el acceso concurrente al sistema por parte de los usuarios y el mantenimiento de sus datos entre sesiones de trabajo del sistema.
- **La evaluación de las facetas de rendimiento y usabilidad** presentadas por el sistema. Los experimentos realizados se replicarán utilizando los mismos scripts empleados por los alumnos en el desarrollo de las prácticas de laboratorio para contrastar el beneficio conseguido.

## 1.3. Metodología de trabajo

El trabajo realizado tanto en este proyecto se ajusta a la línea de trabajo del método de la ingeniera, que supone un paradigma evolutivo en el que, a base de iteraciones, se obtienen sistemas eficientes que resuelven los diferentes problemas. Este método se estructura en las etapas de observación de soluciones ya existentes, propuestas de soluciones mejores a las anteriores, desarrollo de la propuesta, medición y análisis de la solución obtenida y la repetición del mismo proceso hasta que no pueda mejorarse aún más [Adr93].

La solución en la que se basa este proyecto es la presentada en [SI16], que, según se ha comentado con anterioridad, ofrece margen de mejora en su funcionamiento interno. Una vez se han revisado los problemas presentes en dicha versión y las funciones pendientes de integrar, se han propuesto las diferentes mejoras que serán expuestas a lo largo de este documento.

Para el desarrollo de estas propuestas se sigue un proceso iterativo donde se trabaja en un subconjunto de las propuestas. Al término de estas iteraciones, de dos semanas de duración, se mantiene una reunión con los tutores de este TFM, donde se reportan los logros alcanzados y los problemas detectados en las semanas de trabajo previo con la redacción de informes de seguimiento. Al finalizar cada reunión se determina la línea de trabajo a seguir en la siguiente iteración.

Cuando se han completado las diferentes mejoras acordadas para la presente versión del DNSE3, se ha analizado su comportamiento por medio de pruebas sintéticas en un entorno seguro para la medición de su rendimiento y con la colaboración de usuarios reales para la usabilidad de la aplicación.

Al término de este proyecto se han agrupado los diferentes problemas detectados a lo largo de las diferentes pruebas que serán revisadas en futuros trabajos sobre el sistema.

## 1.4. Estructura del documento

La propuesta del DNSE3 mostrada a lo largo de este documento ha sido plasmada en dos artículos, incluidos en los apéndices A y B. El primero de ellos ha sido aceptado para su publicación en las Actas de las XIII Jornadas de Ingeniería Telemática de 2017 y está más centrado en las decisiones tomadas para aprovechar el escalado de la nube en la ejecución de las simulaciones de redes. El segundo, pendiente de su envío futuro próximo, está más orientado a su uso dentro de la docencia.

Aunque en ambos artículos se presenta el trabajo realizado a lo largo de este TFM, en este documento se amplía esta información, revisando las diferentes decisiones de diseño tomadas en el trabajo y analizando con mayor detalle los resultados mostrados en los artículos.

En lo que resta de documento se ilustrará el trabajo realizado sobre el DNSE3. En el capitulo 2 se revisarán los rediseños que ha sufrido el DNSE3, tanto en sus funciones internas como en aquellos sistemas que afectasen directamente a su escalado.

Con el sistema ya operativo, en el capitulo 3 se recogerán los resultados obtenidos en los diferentes experimentos realizados sobre él, con el propósito de evaluar su comportamiento en comparación con el uso del ns-3 mediante *scripts* de línea de comandos, según se ha realizado en las prácticas de laboratorio en las asignaturas de teletráfico. Estas pruebas analizarán el rendimiento medido por medio de los tiempos de respuesta ante diferentes cargas de trabajo, así como la usabilidad que perciben los usuarios a través del cliente gráfico.

Finalmente, en el capítulo 4 se recogen las principales conclusiones obtenidas al término de este proyecto, acompañadas de las propuestas de líneas de trabajo futuro a seguir con el sistema.

# Capítulo 2

# Revisión y rediseño del sistema

En la introducción de este documento se hizo patente la presencia de diferentes problemas en el prototipo del DNSE3 que impedían su consumo por parte de los usuarios finales, siendo una de las más notables la poca madurez presente en el manejo del entorno multiusuario. La falta de persistencia de los datos de los usuarios no presentes en los ficheros almacenados en el servicio de almacenamiento, como los parámetros utilizados en las simulaciones o su evolución tras la parada del sistema; la pendiente integración de un sistema de autenticación y autorización para el acceso de los usuarios o la falta de una interfaz gráfica fueron algunas de las carencias detalladas en las lineas de trabajo futuras en [SI16]. Pero no se quedaba ahí la cosa. En las pruebas de viabilidad del sistema se pudieron experimentar otra serie de problemas que mermaban el rendimiento ofrecido, como es la colisión de diferentes instancias de simulación al intentar acceder a la misma tarea de simulación [SI16].

En el presente capitulo se abordan todos estos problemas y se proponen una serie de cambios que ayuden a solventarlos, tanto en un rediseño en su funcionamiento como una revisión de su implementación. Así, en la sección 2.1 se detallarán las causas y rediseños que han sufrido tanto en su arquitctura como en los servicios involucrados. La sección 2.2, por otra parte, se enfocará en las modificaciones que afectan a los mecanismos y sistemas relacionados con el escalado de los servicios de simulación, es decir, el reparto y asignación de las tareas de simulación y las políticas de escalado. En la sección 2.3 se presentará el cliente gráfico diseñado para el acceso y uso del sistema por parte de los alumnos de las asignaturas de teletráfico para concluir, finalmente, en la seccion 3.1 con la infraestructura desplegada para el uso de su versión final.

## 2.1. Modificaciones en el diseño del sistema

Según se indicó en la introducción, la antigua versión del DNSE3 presentaba ciertas carencias que dificultaban en mayor medida su uso por parte de usuarios reales, entre las que se encuentran la falta de persistencia de los datos de los usuarios no respaldados en el servicio de almacenamiento y la falta de un sistema de gestión de la autenticación y autorización de las peticiones de los usuarios. No obstante, hay otros aspectos que ofrecen un margen de mejora para mejorar la respuesta ofrecida. Tal es el caso del ciclo de trabajo del servicio de simulación o la publicación de nuevos proyectos.

En esta sección se revisarán cada uno de los cambios introducidos en la aplicación, agrupándolos por su efecto en la arquitectura y en cada uno de los servicios que la componen.

### 2.1.1. Arquitectura del sistema

De los diferentes componentes que han sido revisados en el ejercicio de este proyecto, la arquitectura de la aplicación ha sido la menos afectada. Sigue siendo orientada a servicios (*Service Oriented Architecture*, SOA), donde cada una de las funciones a desempeñar por el sistema se han segregado en servicios independientes a los que acceder por medio de sus interfaces, que facilia las labores de desarrollo y mantenimiento del sistema, ya sea por la evaluación aislada de los diferentes servicios o por su capacidad de continuar con su funcionamiento a pesar de que alguno de ellos falle [Vin07].

De la misma forma, cada uno de los servicios manitenen su diseño RESTful, ofreciendo una API (*Application Public Interface*) REST (*REpresenational State Transfer*) para el acceso a sus funciones a traves de los recursos expuestos [Ric08]. Entre las características de los servicios RESTful, la que más valor añade a la aplicación es la interacción sin estado con los recursos. Cada vez que se realiza una petición al servicio, el responsable de proporcionar y mantener el estado de la aplicación es el cliente en lugar del servidor, quien sólo se encargará de mantener el estado de sus recursos internos. De esta forma, el servidor no se preocupa por el número de clientes que tiene actualmente activos, con lo que se simplifican tanto los datos que gestiona como su replicación para escalar el sistema, requiriendo únicamente el acceso a los datos originales [Ric08].

Donde sí que ha sufrido alguna alteración la arquitectura es en los servicios que la conforman, con la eliminación de algunos y la modificación interna de los que se han conservado. En el siguiente listado se presentan los servicios que conforman la actual arquitectura, al igual que en la figura 2.1, que muesta además las interacciones establecidas entre ellos.

- **Servicios genéricos de la aplicación** (SGA): Estos servicios presentan un bajo nivel de acoplamiento con el sistema, dada la generalidad de las funciones que realizan, pero que no están vinculadas con el entorno de nube computacional. Todos estos servicios pueden ser facilmente adaptados para su uso en otras aplicaciones que requieran de sus funciones. Dentro de esta categoría se pueden encontrar:

  - **Servicio de colas** (SGA-01): Este servicio mantiene y gestiona un sistema de colas en la que se almacenan tareas de carácter general para que sean procesadas y ejecutadas de forma asíncrona por agentes externos. En el caso del DNSE3, estas tareas abarcan tanto las simulaciones generadas por los usuarios, tanto a partir de las simulaciones individuales como de barrido de parámetros, y los trabajos de infrome que agrupan los resultados de las simulaciones anteriores.
  - **Servicio de informes** (SGA-02): Este servicio es el encargado de recoger y agrupar los resultados originados tras la ejecución de los modelos de simulación. En los ficheros generados se clasifica cada una de las variables que pre-

sentan los diferentes resultados en función de las combinaciones de parámetros empleadas en la ejecución de las simulaciones.

- **Servicios propios de la aplicación** (SPA): Cada uno de estos servicios está fuertemente acoplado con el DNSE3, al ofrecer una serie de funciones muy concretas y formar parte del núcleo de la aplicación. Aunque pueden acondicionarse para su uso en otros sistemas, el trabajo necesario para dicho fin es bastante elevado. Dentro de esta categoría se pueden encontrar:
  - **Servicio de orquestación** (SPA-01): Este servicio es el encargado de coordinar las diferentes tareas a desempeñar por el resto de servicios implicados. Con este fin, este servicio supone el punto de acceso al sistema por parte de los usuarios finales y distribuye las peticiones recibidas entre el resto de servicios. De forma específica, genera cada una de las tareas de simulación y de informe que son publicadas en el servicio de colas.
  - **Servicio de simulación** (SPA-02): Este servicio es el encargado de realizar la ejecución de los modelos de simulación del ns-3 según se dictamina en las tareas recogidas desde el servicio de colas, a quien notificará del éxito o no en su procesamiento. Al término de estas ejecuciones deberá almacenar los resultados obtenidos en el servicio de almacenamiento.
- **Servicios propios de la infraestructura** (SPI): Estos servicios presentan un alto acoplamiento, más que con la aplicación, con el entorno de la nube computacional. Están especialmente enfocados para la gestión y administración de los recursos ofrecidos por la infraestructura. Dado que la gran mayoría de *middlewares* de gestión de la nube implementan sus propias soluciones, adaptadas al propio entorno que gestionan, se ha optado por acondicionar su uso dentro de las necesidades del DNSE3. Dentro de esta categoría se pueden encontrar:
  - **Servicio de almacenamiento** (SPI-01): Este servicio ofrece un repositorio de ficheros de propósito general de acceso remoto y autoescalable. Debrá permitir la realización de operaciones CRUD (*Create, Read, Update and Delete*) sobre los modelos de simulación agregados por los usuarios, los ficheros de resultados generados tras las ejecuciones individuales de las simulaciones y los informes generados tras la agrupación de los resultados.
  - **Servicio de monitorización** (SPI-02): Este servicio gestiona aquellas métricas que muestran el uso que se hace del sistema en tiempo real. Estas métricas se utilizarán para la toma de decisiones por parte del servicio de escalado.
  - **Servicio de escalado** (SPI-03): Este servicio se encarga de adecuar los recursos empleados por la propia aplicación basandose en la carga de trabajo pendiente de procesar, utilizando una serie de políticas de escalado que revisan las métricas presentes en el servicio de monitorización.

Si se contrasta la actual propuesta con la presentada en [SI16] se puede apreciar la desaparición de dos servicios: el servicio de estadística, capaz de efectuar diferentes operaciones y métodos estadísticos para la obtención de estadísticas relacionadas con los datos procedentes de la ejecución de las simulaciones, y el servicio de gestión de sesiones,

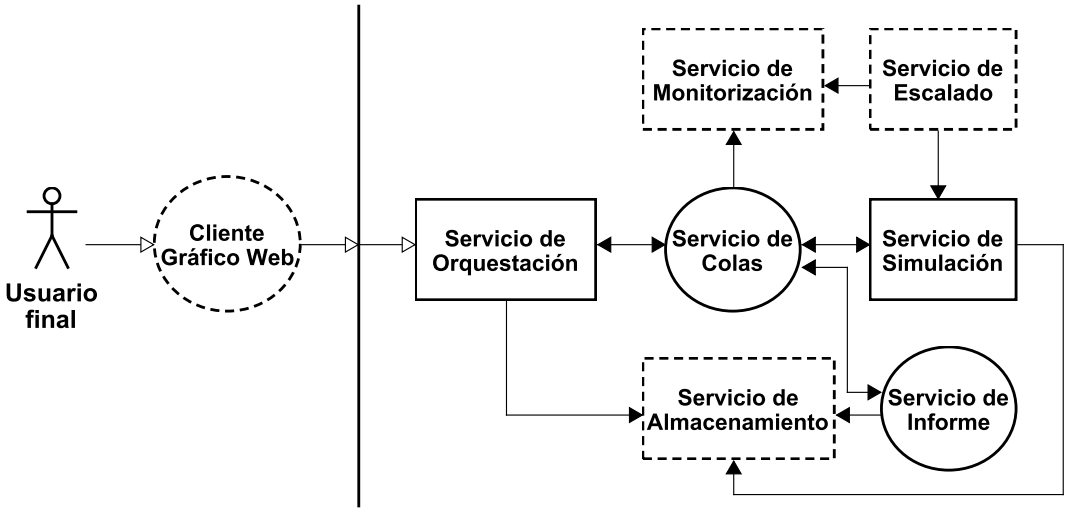


Figura 2.1: Arquitectura orientada a servicios del DNSE3. Las cajas con trazo discontinuo representan los servicios propios de la infraestructura. Las circunferencias con trazo continuo son los servicios genéricos de la aplicación que pueden ser fácilmente extendidos a otras aplicaciones y entornos, mientras que las cajas con trazo continuo son los servicios propios de la aplicación

que mantiene y gestiona el listado de usuarios registrados en la aplicación y sus sesiones de trabajo.

El primero de estos dos servicios ha sido descartado temporalmente de la arquitectura al no ser un servicio fundamental para el buen funcionamiento del sistema. Para la revisión de los resultados obtenidos en la ejecución de las simulaciones se requiere actualmente del uso de herramientas de terceros, como son MATLAB[1], GNU Octave[2] o Microsoft Excel de la suite Office[3], que de uso habitual por los principales usuarios objeto del sistema. No obstante, se preve en el futuro presentar una solución integrada dentro del sistema que facilite esta labor por medio de la interacción dinámica con el usuario final.

El servicio de gestión de sesiones, por su parte, se ha retirado de la arquitectura del sistema a favor de la interoperación con sistemas externos de autenticación de los usuarios. Por lo general, los entornos a los que está orientado el DNSE3 suelen contar con otra serie de herramientas que disponen de unas cuentas de acceso. Con el acceso a los sistemas de autenticación externos por parte del DNSE3 se ofrece su acceso a través de las mismas credenciales que ya utilizan. En esta versión se ha integrado el servidor LDAP (*Lightweight Directory Access Protocol*) gestionado por la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid.

[1] `https://es.mathworks.com/products/matlab.html`
[2] `https://www.gnu.org/software/octave/`
[3] `https://www.office.com/`

### 2.1.2. Servicio de orquestación

El servicio de orquestación es, dentro de los servicios desarrollados para el sistema, el más complejo de todos ellos dada la gran cantidad de funciones que debe desempeñar. Al ser el único servicio sobre el que los usuarios pueden realizar peticiones, la mayoría de sus funciones están orientadas a la orquestación del sistema, es decir, coordinación del resto de servicios para atender a estas peticiones. Por la gran variedad de tareas que debe realizar, no es de extrañar que sea también el servicio que más problemas y carencias presentara en su implementación. En la antigua versión del servicio se realizó un especial esfuerzo en aquellas partes que mayor implicación tenían con el uso final del sistema, es decir, la ejecución de las simulaciones y se dejaron de lado otras funciones para facilitar su uso por los usuarios, como son la persistencia de los datos de los usuarios o el proceso de creación de los proyectos.

En aquella versión no se prestó atención al almacenamiento de los datos de los usuarios al dejarlo de lado en pos de disponer de las funciones esenciales para la ejecución distribuida de las simulaciones. Los únicos datos que se almacenaban eran los modelos de simulación de los usuarios y los resultados generados tras su ejecución, de forma que los datos relacionados con los parámetros de los modelos, los diferentes ficheros que se obtienen tras su procesado o el estado en el que se encontraba la ejecución de las diferentes simulaciones eran eliminados cada vez que arrancaba el servicio.

Por otro lado, la creación de los proyectos de simulación ha sido otro de los puntos conflictivos de la pasada versión. En esta generación se enviaba al servidor el modelo que sería posteriormente simulado acompañado de un nombre y una breve descripción que ayudase a su identificación posterior. El resto de datos que condicionan la ejecución del modelo, como son sus parámetros de entrada admitidos (junto con sus valores admitidos) y los ficheros generados que contienen los resultados, debían ser indicados a posteriori y de forma manual una vez se creaba una simulación concreta cuando se podría haber incluido en la etapa de creación del proyecto.

Este **respaldo de los datos de los usuarios** se integra mediante el acceso del servicio de orquestación a un gestor de bases de datos que, además de almacenar estos datos en diferentes tablas de la base de datos, permite consultarlos y analizarlos de forma eficiente. Debido a que los servicios del DNSE3 se desarrollan en el lenguaje de programación Java, se usa JPA (*Java Persistence API*) para operar sobre el gestor. Gracias a JPA, en lugar de operar en el nivel de las tablas almacenadas, se abstrae el uso de los gestores por medio de un proveedor de JPA, quien traduce las diferentes operaciones al gestor utilizado y mapea los datos de las tablas en los diferentes atributos de las clases utilizadas. Si en algún momento fuese necesario la migración de la base de datos a otro servidor o cambiar de gestor, esta configuración se puede modificar sin alterar la implementación interna del programa.

Para garantizar la integridad de los datos presentes en las tablas, se accede a la base de datos de forma bloqueante, es decir, solo puede un único hilo de ejecución de forma simultánea a la base de datos. Aunque a priori esta decisión choca con el acceso concurrente de los usuarios al propio servicio, estas operaciones son relativamente ligeras y no se traduce en retrasos notables en las peticiones de los usuarios.

En lo referente a la creación de los proyectos, ésta se ha visto mejorada mediante la inclusión de un fichero XML (*eXtensible Markup Language*) en el envío del modelo de

simulación que, además de incluir el nombre y la descripción como se requería anteriormente, incluye el resto de metadatos que definen el uso del modelo: los parámetros de entrada aceptados; el tipo de valor que toma cada uno de los parámetros (entero, racional, cadena de caracteres, o semilla para la generación de valores aleatorios); su rango de valores admitidos; los ficheros de resultados que genera tras su ejecución; la clase de fichero que son (de resultados, si contienen el valor final de una serie de variables; tabular, cuando ofrece el valor de diferentes variables a medida que ha progresado su ejecución; o de traza, con la captura de trafico en algún punto del modelo), y, en el caso de los ficheros de resultado y tabulares, la correspondencia entre los valores presentes en ellos y las variables que representan. Todos estos datos son presentados siguiendo el esquema XSD (*XML Schema Definition*) presente en el apéndice C.

### 2.1.3. Servicio de simulación

El servicio de simulación, según se ha presentado en la arquitectura del sistema, es el responsable de la correcta ejecución de las simulaciones de los usuarios, donde se incluye el uso de los parámetros de entrada asignados por el usuario y la publicación en el servicio de almacenamiento de los resultados solicitados. Para realizar este cometido mantiene un contacto continuo con el servicio de colas para la recogida de las tareas de simulación. En la pasada versión del DNSE3 se seguía el siguiente ciclo de operación:

1. El servicio de simulación realiza una solicitud HTTP (*HyperText Transfer Protocol*) GET al servicio de colas para **consultar** el recurso asociado a **la siguiente tarea pendiente de ejecutar**. El servicio de colas revisa su cola de tareas FIFO (*First In, First Out*) para recoger o bien la primera tarea ya en ejecución que ha sido abandonada por su trabajador o bien la primera tarea disponible.

2. **El servicio de simulación revisará que la tarea es válida** consultando la presencia de su modelo dentro del servicio de almacenamiento para garantizar que el proyecto asociado sigue existiendo dentro del sistema.

3. Tras esta comprobación, **el servicio de simulación reserva dicha tarea** para que no pueda ser utilizada por otra instancia. Esta reserva se efectúa por medio de una petición HTTP PATCH donde se incluye, además de la dirección IP del propio trabajador, el valor de HTTP ETag [Fie14] asociado al estado previo de la tarea[4].

4. Con la tarea ya a su disposición, **el servicio de simulación comienza la ejecución** formal de la tarea, compilando el modelo recogido del servicio de almacenamiento y procediendo con su posterior procesado. Mientras tanto, de forma paralela, envía periódicamente peticiones HTTP PATCH para notificar al servicio de colas que continúa con la ejecución de la tarea para evitar su asignación a otra instancia del servicio.

[4]Cada una de las tareas presentes en la cola tienen asociado una etiqueta HTTP ETag para su estado actual. Si se desea modificar alguno de los datos de la tarea, se debe adjuntar a la petición el valor de esta etiqueta para garantizar que se altera la versión indicada y, una vez completada la modificación, se cambiará por un valor generado aleatoriamente. Esta etiqueta garantiza que dos instancias del servicio de simulación intenten ejecutar la misma tarea, de tal forma que se rechacen sus peticiones una vez no coincida el valor de la etiqueta.

5. Una vez completada su ejecución y preservado los resultados generados dentro del servicio de almacenamiento, se envía una última petición HTTP PATCH donde indica este éxito y pueda volver al primer paso de este ciclo.

El problema más destacable dentro de este flujo de operaciones es **la necesidad de compilar el mismo modelo** en cada una de las ocasiones que este es procesado. Aun suponiendo que este tiempo sea relativamente corto para la gran mayoría de los modelos, la repetición continua de esta compilación se incrementa el tiempo total de respuesta una cantidad no despreciable, especialmente cuando el número de simulaciones es bastante elevado.

Para minimizar este efecto, se ha trasladado la compilación del modelo al servicio de orquestación, quien lo realizará mientras se crea el proyecto asociado al modelo. Además de disponer de una versión del modelo directamente utilizable para sus futuras ejecuciones, este cambio permite notificar al usuario de los posibles fallos presentes en el modelo sin tener que esperar a que se lance una simulación.

El siguiente cambio introducido dentro de este servicio tiene como objetivo **la explotación de la paralelización *hardware* para la ejecución de las simulaciones**. Hasta ahora, el flujo presentado anteriormente se ejecutaba de forma secuencial en un único hilo de ejecución, pero puede ser separado en dos hilos diferentes: uno encargado de la recogida de las tareas y el otro de la ejecución de estas. El principal beneficio que aporta esta división es la posibilidad de lanzar múltiples hilos que se encarguen de la ejecución de las simulaciones, de tal forma que cada servicio de simulación pueda trabajar con más de una tarea a la vez. Será necesario comprobar con anterioridad el número de hilos óptimo soportado por la instacioa en la que se despliega el servicio.

Los hilos de recogida de tareas y de ejecución se comunican a través de una pila de tareas, cuyo tamaño será igual al número máximo de hilos de ejecución simultáneos de simulación soportados por el servicio ($N_{jobs}$). Cuando el hilo de recogida de tareas consiga recuperar una del servicio de colas, la publicará dentro de la pila de tareas interna para que sea recogida inmediatamente por un hilo de ejecución. Esta tarea se mantendrá dentro de la pila hasta que se haya finalizado su ejecución. Si la pila se encuentra llena, es decir, el servicio tiene actualmente $N_{jobs}$ hilos de simulación activos, el hilo de recogida de tareas se mantendrá suspendido hasta que alguno de los otros hilos finalice su trabajo y libere un hueco dentro de la pila.

### 2.1.4. Servicio de informes

El servicio de informes fue el que mostraba un menor nivel de madurez al término del sistema anterior. Este servicio presentaba una cola interna, semejante a la presente en el servicio de colas, donde el servicio de orquestación enviaba las peticiones de informes una vez se finalizaba cada una de las simulaciones. Con estas peticiones, este servicio contactaba con el servicio de almacenamiento para recoger los diferentes resultados obtenidos durante la ejecución y los agrupaba en las diferentes combinaciones de parámetros utilizadas. Para saber a qué combinación estaba asociado cada uno de los datos se consultaba una parte del URI (*Universal Resource Identifier*) que mostraba sus valores.

En la versión actual del servicio se ha revisado su funcionamiento y modificado su implementación, siendo ahora una iteración sobre el servicio de simulación. Al separar

los hilos de recogida de tareas y de ejecución, se puede modificar fácilmente este último para incorporar un sistema que procese una serie de tareas recogidas de una cola. La cola anteriormente gestionada por el servicio se ha trasladado al servicio de colas y es totalmente independiente de la de simulación. La ventaja principal que ofrece este cambio de implementación es el acceso a las mismas ventajas presentes en el servicio de simulación de forma externa, como es el escalado del servicio, en caso de considerarse necesario.

La **generación de los informes** también se ha visto mejorada en esta nueva versión. Los resultados de las simulaciones se almacenan dentro de diferentes ficheros CSV (*Comma Separated Values*) que pueden ser utilizados directamente por herramientas de terceros y cuya estructura facilita la localización de los diferentes experimentos realizados. Entre estos informes se encuentran los asociados a los parámetros de entrada de la simulación, que muestran a lo largo de una columna los diferentes valores que ha tomado en la ejecución el parámetro indicado en el nombre del fichero, y a los resultados de la ejecución, cuyo nombre indica el fichero de donde se ha extraído el resultado y el nombre de este último (`nombreFichero_nombreResultado.csv`). Cada una de las filas de estos ficheros se corresponde con cada una de las diferentes combinaciones de parámetros que se han utilizado y, en los ficheros de los resultados, cada una de las columnas está asociada a las diversas repeticiones de cada combinación. En el apéndice D se muestra el contenido de unos ficheros de ejemplo. Para conocer la relación que hay entre los diferentes parámetros y los resultados generados en las diferentes ejecuciones, el servicio de simulación añade junto a los ficheros de resultados otro fichero con notación JSON (*JavaScript Object Notation*) que contiene el valor de los diferentes parámetros utilizados y que será consultado posteriormente por el servicio de informes para su correcta agrupación.

## 2.2. Rediseño de los mecanismos implicados en el escalado

El motivo principal que incentivó la migración del sistema a una infraestructura de nube es el escalado automático de los recursos utilizados. Los *middlewares* de gestión de la nube registran una serie de métricas que representan el uso que se hace de los recursos, como son el tiempo que llevan activas las instancias o el uso de CPU virtual. Sin embargo, en ocasiones estas métricas no describen la carga real del sistema, por lo que resulta necesario la generación de métricas personalizadas. De la misma forma, estos sistemas describen la sintáxis a seguir en la definición de las reglas de escalado pero en ningún momento incorporan alguna regla que pueda ser empleada.

Para sacar partido de este escalado es necesario definir una métricas de aplicación pertinentes así como unas reglas diseñadas cuidadosamente para balancear un escalado rápido que minimice el tiempo de respuesta con un uso ajustado de los recursos.

La pasada versión del DNSE3 ya ofrecía una solución para todos estos asuntos. Recogía en una métrica propia el número de tareas pendientes de ejecutar y utilizaba unas reglas de escalado propias para comprobar el buen funcionamiento del sistema. Además, incorporaba un sistema de asignación de estas tareas entre los nodos disponibles con la delegación del rol activo de asignación siguiendo el modelo *work-stealing* [Blu99], de forma que sean los propios nodos quienes se encarguen activamente de recoger la siguiente

tarea.

Sin embargo, el sistema de escalado presentaba ciertas deficiencias que impedían su uso en un entorno real. Las reglas de escalados estaban orientadas a la evaluación del sistema de escalado más que al uso responsable de los recursos. Por su parte, el reparto de las tareas entre las instancias de simulación, a pesar de funcionar correctamente, mostraba de forma frecuente colisiones entre éstas al intentar acceder a la misma tarea de simulación.

En este proyecto se revisan estos dos sistemas para garantizar un uso responsable y eficiente de los recursos de la nube y para ofrecer el mejor rendimiento posible durante su uso habitual. En primer lugar se centrará este rediseño en la asignación de las tareas de simulación y, posteriormente, se presentará una nueva política de escalado que no incurra en un derroche de recursos.

### 2.2.1. Reparto y planificación de las tareas de simulación

En un entorno distribuido que no ofrezca un escalado automático de sus nodos de computación, se puede configurar con sencillez al servicio encargado de distribuir cada uno de los trabajos. Se le puede indicar qué nodos tiene a su disposición, acompañados de sus respectivas direcciones IP para el envío de los trabajos y mantener un contacto continuo para conocer la evolución de su ejecución. Sin embargo, cuando el número de estas instancias va aumentando a lo largo del tiempo, resulta inviable el mantenimiento de un registro de todos ellos sin incurrir en pérdidas de rendimiento cuando su número es lo suficientemente elevado.

La forma en la que el DNSE3 aligeraba este seguimiento del sistema consistió en un cambio del rol activo en el reparto. Según establece el modelo *work-stealing* [Blu99], serán los propios trabajadores (en este caso, las instancias de simulación) quienes se preocupen de recoger la siguiente tarea a procesar. Este flujo de operaciones es el mismo que se presentó en la sección 2.1.3.

Aunque este diseño es correcto y se ha comprobado la denegación de la reserva si la tarea en cuestión ya está siendo ejecutada, las pruebas de evaluación del sistema completo han mostrado **el acceso simultáneo de las instancias a la misma tarea**. Tal y como se ha indicado anteriormente, una tarea no está en propiedad de un trabajador hasta que no se efectúa la petición de reserva y, dado el comportamiento asíncrono de las instancias de simulación, existe una ventana temporal entre la solicitud de la siguiente tarea y su reserva donde otras instancias de simulación preguntan también por la siguiente. En la anterior versión del DNSE3, como la tarea siguiente seguía siendo la misma al no perder su posición en la cola, se proporcionaba a todos estos trabajadores la misma tarea y solo notarían que ya estaba siendo ejecutada cuando se descartase la solicitud de reserva. Aunque esta colisión no abarcaba más que unos breves instantes, su repetición continua a lo largo de una carga de trabajo elevada incrementaba de manera apreciable el tiempo de respuesta final.

Otro problema que presenta la cola única de tareas es **el posible acaparamiento de los recursos de simulación por parte de algún usuario**, ya sea de forma accidental o totalmente intencionada. En esta cola se van agregando al final las tareas generadas por todos los usuarios del sistema, de tal forma que si un usuario solicita una gran cantidad de simulaciones, estas tendrán preferencia sobre aquellas generadas posteriormente por otros usuarios y deberán esperar a que se complete la ejecución de todas las anteriores.

Para abordar ambas deficiencias, en este trabajo se ha modificado el proceso de determinación de la siguiente tarea disponible mediante un sistema de rotación *round-robin*. Cada vez que un trabajador solicita la siguiente tarea presente en la cola, esta se verá desplazada al final de la cola para garantizar que la siguiente petición reciba una tarea completamente diferente. Esta retirada anticipada de la cola evita la colisión de las instancias de simulación al disponer cada una de ellas una tarea completamente diferentes, siempre y cuando se disponga de un número suficiente de tareas pendientes.

Esta rotación se realiza en dos niveles, el primero de ellos vinculado al usuario propietario de las tareas y el segundo de ellos ya dentro del listado de las tareas de cada usuario. En la figura 2.2 se muestra el funcionamiento de estos dos niveles. De la misma forma que se ha comentado anteriormente, estos niveles garantizan que dos tareas consecutivas no sean de la autoría del mismo usuario y tengan todos el mismo derecho a consumir los recursos de simulación, siempre y cuando los trabajos pendientes pertenezcan a usuarios diferentes.

El problema que presenta este rediseño está relacionado con la ejecución de múltiples simulaciones por parte de un mismo usuario. Cada usuario dispone de una única cola de tareas de simulación que se va llenando a medida que publica nuevas tareas. Si un usuario solicita la ejecución de una simulación de apenas unas pocas decenas de tareas individuales después de haber arrancado una simulación de un tamaño mucho mayor y no ponga en pausa ninguna de ambas, deberá esperar a que se complete esta simulación tan larga para poder empezar con la otra simulación. Una forma de evitar esta situación sería mediante la agregación de un nuevo nivel intermedio dentro de la cola de trabajo que permitiera conocer a qué grupo de simulaciones pertenece cada tarea, de forma que garantice que las dos tareas ofrecidas tras las peticiones consecutivas al siguiente trabajo de simulación no pertenezcan al mismo grupo.

### 2.2.2. Reglas de escalado de los servicios de simulación

Hasta ahora, el prototipo presentado del DNSE3 mostraba cómo la propia infraestructura de la nube era capaz de responder a las métricas generadas por la propia aplicación y modificar los recursos de simulación según una sencilla regla que tenía en cuenta estos valores. La regla cuyo propósito era exclusivamente la verificación de este sistema, que consistía en agregar una nueva instancia siempre que existiera un trabajo pendiente o todavía en ejecución dentro de la cola. Cada vez que se revisaba esta regla, ya sea tras el periodo de evaluación regular fijado en el servicio de escalado o bien después de haber modificado el número de instancias, se generaba una instancia adicional que ayudaba con la ejecución de las simulaciones.

Esta política tan sencilla presenta un comportamiento muy agresivo, ya que no considera la carga efectiva de trabajo pendiente para el lanzamiento de nuevas instancias. Simplemente verifica que hay tareas pendientes y genera más y más recursos, originando un entorno extremadamente sobredimensionado que solo se reduciría cuando no quedara ninguna tarea por procesar.

Es por ello que se propone la búsqueda de unas reglas de escalado que faciliten, realmente, el acomodamiento del número de instancias a la carga real del sistema. Esta nueva política deberá hacer un uso eficiente de los recursos, evitando continuas fluctuaciones en el número de instancias utilizadas cuando las tareas pendientes esten oscilando cerca

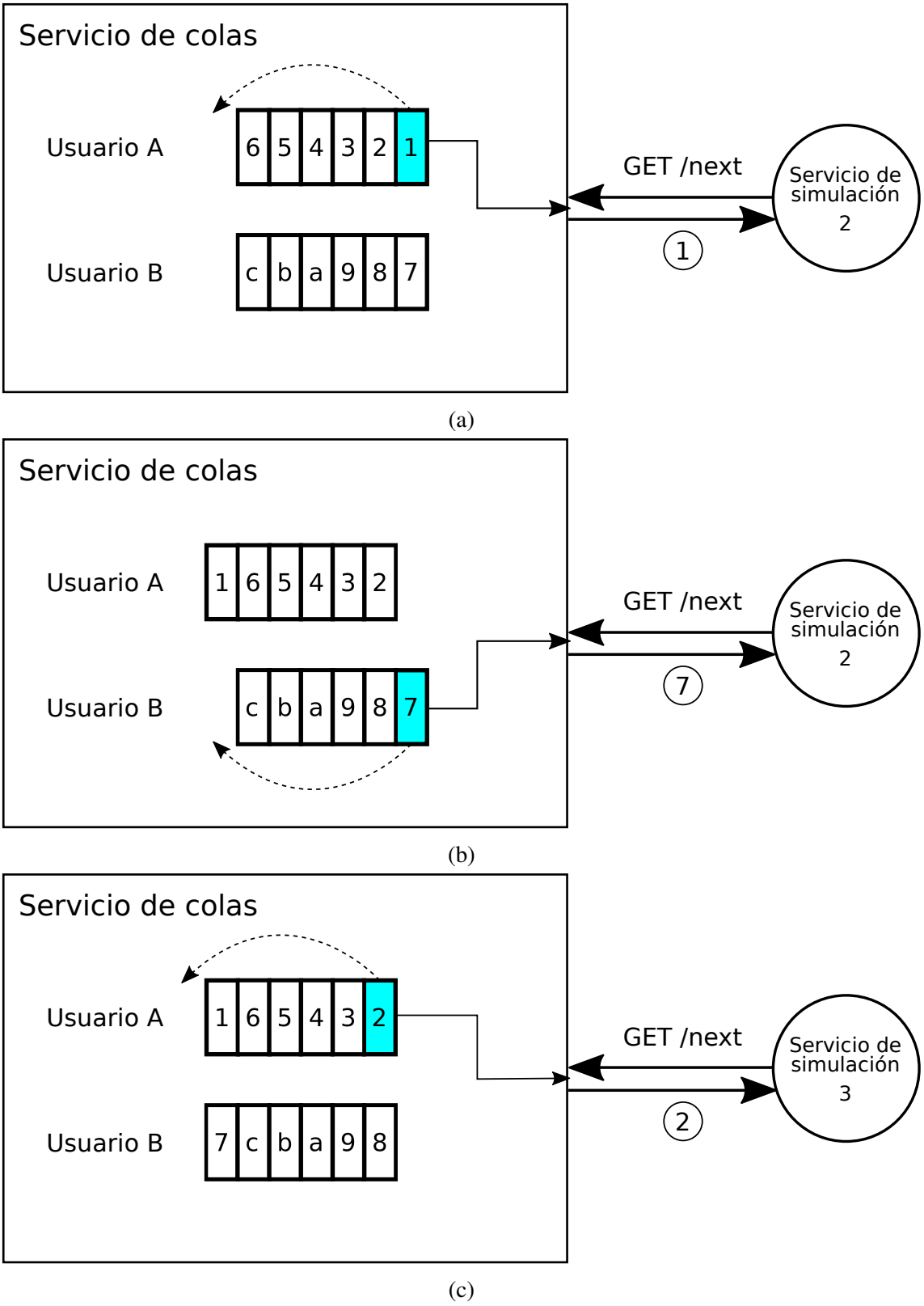


Figura 2.2: Sistema de rotación *round-robin* presente en el DNSE3. Tras recibir la primera petición en (a), la tarea devuelta se desplaza al final de la cola de tareas del usuario. La siguiente tarea devuelta para la siguiente petición será la primera presente en la cola del siguiente usuario, según se muestra en (b). Cuando llega la siguiente petición en (c), al no haber tareas de otros usuarios, se recoge la primera tarea disponible del primer usuario.

de algún umbral de decisión y, por otro lado, mantener la mayor ocupación de recursos posible sin caer en el sobredimensionado escesivo patente en la política anterior.

La propuesta de las reglas presentadas en este TFM se basan en el volumen de tareas pendientes para determinar la carga de trabajo. Para determinar el comportamiento de la regla se mide la proporción de tareas que debe procesar cada una de las instancias actualmente disponibles si se reparten de forma equitativa entre ellas, de tal forma que:

- Para escalar en sentido ascendente, `si` $\frac{tareas}{instancias} > T$ `entonces` $instancias := instancias + 1$.
- Para escalar en sentido descendente, `si` $\frac{tareas}{instancias} < \frac{T}{2}$ `entonces` $instancias := instancias/2$.
- El número de instancias estará siempre comprendido dentro del rango $[minVM, maxVM]$.

Para la obtención de la métrica $\frac{tareas}{instancias}$ se necesita la colaboración del servicio de colas, quien conoce el número total de tareas pendientes, y del servicio de escalado, que se encarga de la gestión de las instancias desplegadas y conoce cuántas se encuentran disponibles. Por otro lado, los parámetros $T$, $minVM$ y $maxVM$ se han expresado sin ofrecer un valor determinado para que el administrador del sistema asigne los valores que estime apropiados para la carga de trabajo esperada para el sistema.

En la figura 2.3 se ilustra el comportamiento de estas reglas, donde se ha asignado el valor $T = 10\frac{tareas}{instancia}$. A la vista de la figura se puede apreciar que los umbrales de la generación y eliminanción muestran una separación apreciable, a excepción del umbral presente entre una y dos instancias activas. Esta separación, junto con los tiempos de reposo entre las diferentes revisiones de las métricas, ofrecen una estabilidad en las instancias desplegadas, al hacer que su número no fluctúe excesivamente cuando lo haría si estos coincidieran y el valor de la métrica oscilara en torno a ellos.

Un beneficio adicional que ofrece esta política es la mejoría en los tiempos de respuesta de las simulaciones. Como las instancias no son eliminadas hasta que la carga de trabajo disminuye de forma sustancial, se dispone de un sobredimensionado temporal de los recursos que ayudan a completar el resto de tareas y a ofrecer un mejor tiempo de reacción cuando lleguen nuevas cargas de tareas.

Esta propuesta de política de escalado no está exenta de limitaciones. La primera de ellas es el arranque conservador que presenta. Aunque la eliminación de las instancias no se produce hasta que haya decrecido notablemente, el número de instancias eliminadas de forma simultánea es mucho mayor que las que se generan posteriormente, incrementándose de una en una, lo que supone un mayor retardo hasta disponer de los suficientes recursos para atender a la demanda de simulaciones. Una posible forma de solucionarlo sería incorporando un crecimiento exponencial de las instancias cuando el valor de la métrica fuese muy superior a $T$, pudiendo duplicar o cuadriplicar el número de instancias activas, o bien insertar de golpe tantas instancias como sean necesarias para que el valor de la métrica $\frac{tares}{instancias}$ sea inmediatamente inferior a $T$.

El otro problema presente en la política es el uso exclusivo del número de tareas pendientes para medir la carga del sistema. Además de este número, hay otros factores que determinan el tamaño de la carga de trabajo, como es la duración requerida para completar cada una de las tareas individuales. Si se supone que la duración de las tareas es

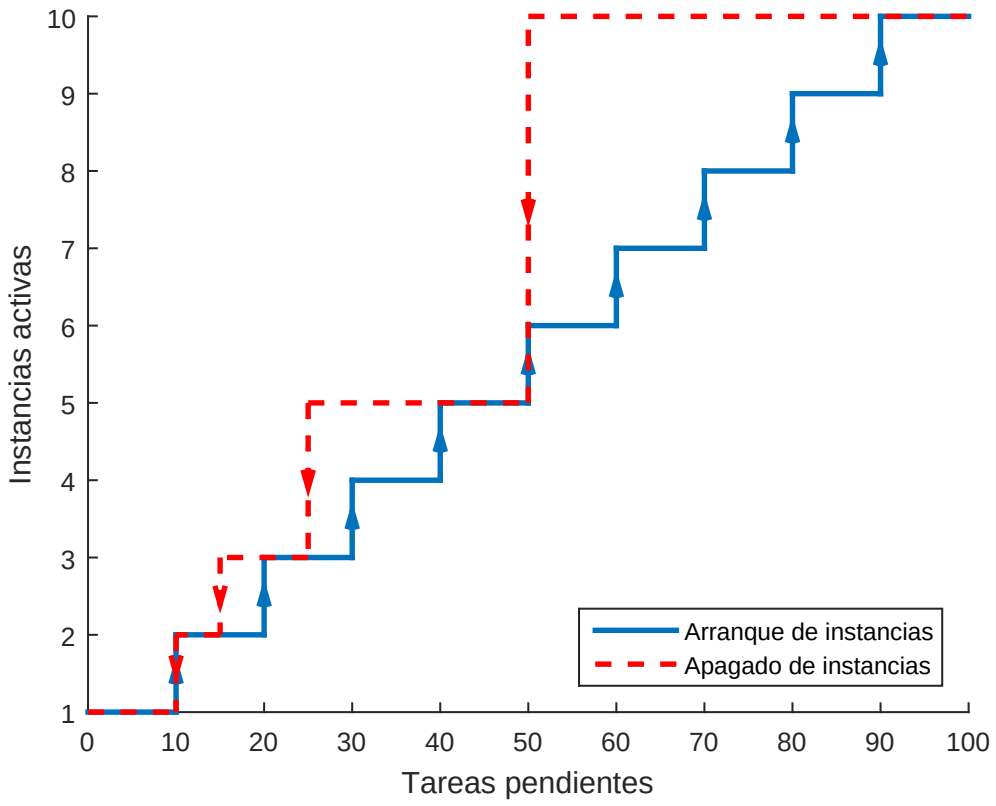


Figura 2.3: Ilustración de las acciones realizadas por las reglas de escalado a medida que varía el número de tareas de la cola, para $T = 10$, $minVM = 1$ y $maxVM = 10$. Las líneas continuas azules representan la generación de nuebas instancias, mientras que en trazo discontinuo rojo, la eliminación de instancias.

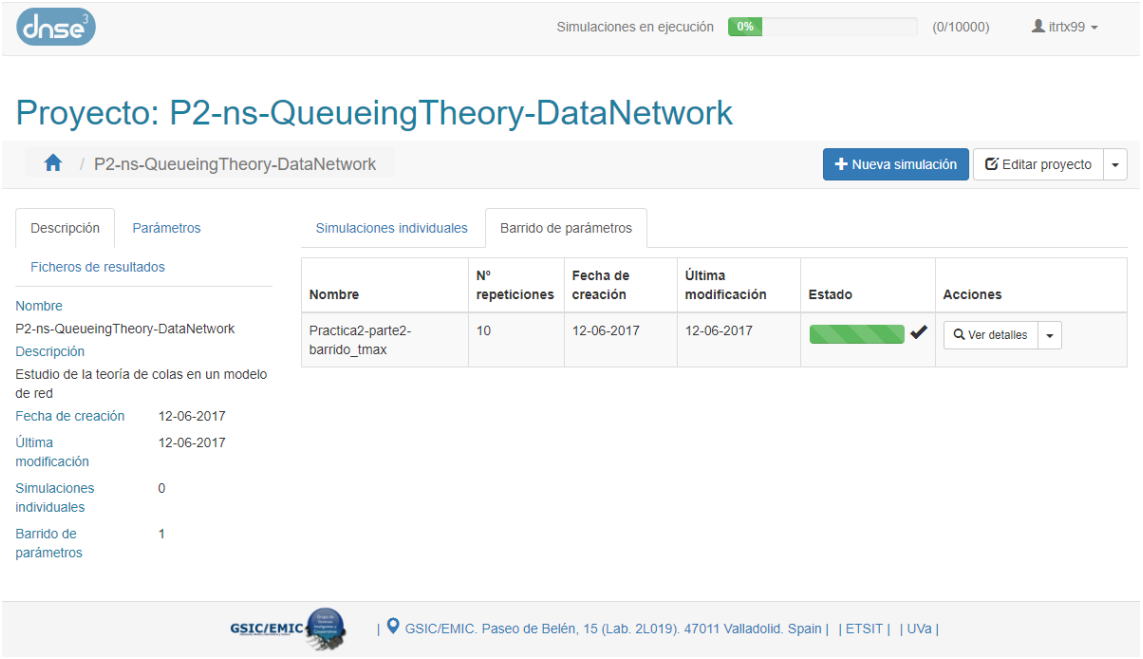


Figura 2.4: Captura de la aplicación gráfica web del DNSE3, donde se ve que la simulación de barrido de parámetros se ha completado correctamente.

razonablemente homogénea, se puede aminorar esta limitación con una adecuada elección del parámetro $T$. Se podría pensar en tener unas reglas de escalado más elaboradas que incluyan el tiempo medio de ejecución de las tareas observado por el propio sistema, de forma que reduzcamos la necesidad de supervisión humana.

## 2.3. Cliente del usuario

El único punto de acceso que ofrece el sistema a los usuarios es el servicio de orquestación, por medio de una API REST compatible con cualquier cliente HTTP para consumir las diferentes representaciones. Durante el desarrollo del entorno se ha empleado un cliente HTTP basado en línea de comandos que permite el uso de *scripts* para comprobar el correcto funcionamiento de cada uno de los servicios, pero es más engorroso de utilizar por los usuarios finales que necesitan comprender el funcionamiento de la API para acceder al sistema. En docencia puede obstaculizar el aprendizaje de los alumnos al pedirles un tiempo y esfuerzo adicional que podrían aprovechar en la realización de las prácticas.

Es por ello que se ha desarrollado una aplicación gráfica web especificamente diseñada para el uso del DNSE3, mostrada en la figura 2.4. Esta aplicación ha sido diseñada por el autor de este trabajo y desarrollada por Javier Enrique Hoyos Torío, miembro del grupo de investigación GSIC/EMIC de la Universidad de Valladolid.

Con esta aplicación se ofrece un acceso completo a las diferentes tareas que pueden realizar los usuarios en el entorno: ver sus diferentes proyectos de simulación, agregar un nuevo proyecto a través de la carga del modelo de simulación según lo expuesto en la sección 2.1.2, ver los detalles de las simulaciones creadas o editarlas, controlar el flujo de ejecución de las simulaciones (iniciarlas, pausarlas, continuarlas o detener por com-

pleto su progreso) y recuperar los resultados finales para su consumo en aplicaciones de terceros que permitan un análisis más exhaustivo. Ha sido desarrollada empleando el framework Bootstrap para ofrecer un acceso móvil al sistema, adaptada tanto a ordenadores personales como *smartphones* y *tablets*.

Para garantizar el acceso seguro a la aplicación se recurre a un sistema de seguridad delegada en el servidor que aloja la aplicación web. El servicio de orquestación está configurado para que sólo confíe en las peticiones procedentes de este servidor, situado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid, y rechazando cualquier otra petición que se genere dentro de la red privada del propio centro.

Esta aplicación se encuentra alojada en uno de los servidores de la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid, ofreciendo un acceso remoto fuera del propio centro. Además de comunicarse con el servicio de orquestación, establece una conexión con el servidor LDAP de la escuela para verificar las credenciales de acceso de los alumnos, coincidiendo con las cuentas empleadas en las sesiones de laboratorio de las asignaturas de teletráfico.

# Capítulo 3

# Evaluación de la versión de producción

Una vez completada la nueva versión del sistema y revisado el funcionamiento de los diferentes servicios implicados, se ha procedido con su evaluación completa para comprobar si ofrece mejoras frente al sistema empleado actualmente en los laboratorios de las asignaturas de Teletráfico, donde se recurre a unos *scripts* que permiten la ejecución secuencial de cada una de las simulaciones con el simulador ns-3.

Para dar muestra del beneficio obtenido con el nuevo entorno, su evaluación se centrará en dos aspectos clave: el rendimiento y la usabilidad del sistema. En primer lugar, en la sección 3.1 se presentará la infraestructura empleada a lo largo de las diferentes pruebas. En la sección 3.2 se evaluará el rendimiento del DNSE3 comparando los tiempos de respuesta obtenidos en ambos entornos. Finalmente, en la sección 3.3 se revisarán las opiniones reportadas por algunos alumnos voluntarios en relación con la experiencia de usuario percibida en los dos sistemas implicados.

## 3.1. Infraestructura del entorno de producción

La infraestructura utilizada en la ejecución de estas pruebas se compone de dos sistemas diferentes. El primero de ellos hace uso de los equipos disponibles en los laboratorios de teletráfico y permiten la evaluación del uso de *scripts* de línea de comandos adaptados para el ns-3, según lo han utilizado los alumnos en las prácticas. Estos equipos disponen de un procesador Intel(R) Core(TM) i5-2400 de 4 núcleos operando a una frecuencia de 3,10 GHz.

La infraestructura utilizada en el DNSE3 se basa en una nube privada, propiedad del grupo de investigación GSIC/EMIC de la Universidad de Valladolid, gestionada con el *software* OpenStack en su versión Mitaka y con acceso a dos nodos de cómputo, el primero con dos procesadores Intel(R) Xeon(TM) E5-2603 de 6 núcleos a 1,60 GHz y el segundo con dos pprocesadores Intel(R) Xeon(TM) E5-2650 de 12 núcleos a 2,2GHz.

La configuración de las instancias utilizadas en los servicios del DNSE3 se distribuye de la siguiente forma:

- El **servicio de orquestación** utiliza una instancia con 2 CPUs virtuales y 4 GB de memoria RAM.

- El **servicio de colas** utiliza una instancia con 2 CPUs virtuales y 4 GB de memoria RAM.
- El **servicio de informes** utiliza una instancia con una CPU virtual y 2 GB de memoria RAM.
- El **servicio de simulación** utiliza una instancia con una CPU virtual y 2 GB de memoria RAM.

Tanto el servicio de orquestación como el servicio de colas disponen de unas instancias de mayor capacidad por la gran cantidad de tráfico entrante prevista en las diferentes pruebas. Por otro lado, el servicio de simulación es el único que dispondrá de varias instancias activas de forma simultánea para ofrecer el escalado en la ejecución de las simulaciones.

En lo que respecta a los servicios propios de la infraestructura, todos ellos emplearán los servicios ofrecidos por OpenStack. Concretamente, se utilizará el servicio de almacenamiento de objetos Swift para el servicio de almacenamiento, el servicio de métricas Ceilometer para el servicio de monitorización y el servicio de orquestación de la nube Heat para el servicio de escalado.

## 3.2. Rendimiento en el tiempo de respuesta

La evaluación del rendimiento del DNSE3 contrasta los tiempos de respuesta obtenidos tras la ejecución de diferentes simulación de barridos de parámetros compuestos por 50, 100, 500, 1000, 5000 y 10000 simulaciones individuales, con tiempos de ejecución homogéneos. Se han empleado estos tamaños de las simulaciones al abarcar trabajos de corta y larga extensión y al asemejarse con las diferentes demandas de los usuarios reales. El modelo de simulación utilizado a lo largo de las pruebas se corresponde con uno de los empleados en las prácticas de las asignaturas de teletráfico, de forma que se asemeje al desarrollo de una práctica real. Para asegurar la estabilidad de los resultados se han repetido 5 veces, a excepción de las simulaciones de 5000 y 10000, que se han repetido 10 veces al presentar una mayor variabilidad en sus resultados.

En el transcurso de las pruebas se ha minimizado la influencia de agentes externos en los diferentes sistemas. En el caso de los equipos de laboratorio se ha asegurado que no hubiera ningún otro proceso de usuario utilizando los recursos de la máquina. En lo que respecta al DNSE3, se ha realizado una pausa entre prueba y prueba de extensión variable y que ha alcanzado la hora para las simulaciones más largas. Esta pausa permite que el sistema parta de una situación de reposo, de forma que tenga que generar de cero los recursos que necesite para atender la demanda de simulaciones y se estabilice la temperatura de la máquina anfitriona por el alto consumo de CPU generado en las instancias de simulación.

Para la política de escalado del sistema se han fijado los umbrales de escalado ascendente ($T$) y descendente ($T/2$) en 10 y 5 tareas por instancia respectivamente. El resto de parámetros que afectan a las reglas de escalado ($minVM$ y $maxVM$) y otros parámetros que afectan al comportamiento del sistema ($N_{jobs}$, número de procesos de simulación

| Parámetro | Valor mínimo | Valor máximo |
|---|---|---|
| $minVM$ | 1 | 5 |
| $maxVM$ | 15 | 30 |
| $N_{jobs}$ | 1 | 4 |

Tabla 3.1: Parámetros utilizados en las diferentes configuraciones del DNSE3 evaluadas

simultáneos en las instancias) han presentado diferentes valores para estudiar el efecto producido en el tiempo de respuesta experimentado, según se muestra en la tabla 3.1.

En la tabla 3.2 se muestran los diferentes resultados obtenidos en las diversas configuraciones. Para revisar estos tiempos se analiza la configuración más conservadora de las presentadas ($minVM = 1, maxVM = 15, N_{jobs} = 1$). En esta configuración se puede apreciar una mejoría destacable cuando la carga de trabajo es muy alta, pero en las más cortas ofrece unos tiempos más largos que en la solución local. Este detrimento en el rendimiento viene propiciado por tanto por las comunicaciones entre los servicios como por el funcionamiento de la política de escalado.

Mientras que una de las simulaciones utilizadas en estas pruebas necesita de apenas 2 o 3 segundos en los equipos de los laboratorios, este tiempo se ve incrementado por los retardos originados por las comunicaciones entre los servicios. Se necesitan 7 segundos para que el servicio de orquestación publique el trabajo en el servicio de colas, el servicio de simulación recoja la tarea. la reserve, recoja el modelo ya compilado del servicio de almacenamiento, complete la ejecución, almacene los resultados de nuevo en el servicio de almacenamiento, notifique al servicio de colas del éxito en la simulación y este último notifique al servicio de orquestación. En el futuro, se podría pensar en mecanismos para que el sistema determine automáticamente cuál es la aproximación más adecuada para hacer el escalado.

Aunque este retardo está presente siempre, sus efectos son rápidamente eclipsados cuando se dispone de suficientes recursos que exploten la ejecución distribuida de las simulaciones. El factor que más afecta a estos resultados es el retardo en el aprovisionamiento de las nuevas instancias. Cada una de las reglas revisa las métricas cada 15 segundos y, una vez se decida la generación de una nueva instancia, esta debe completar su secuencia de arranque para empezar a recoger tareas. Durante este tiempo, el DNSE3 sigue operando con las mismas instancias que disponía anteriormente, por lo que no puede aprovecharlos a tiempo para minimizar el tiempo de respuesta.

Para mejorar esta situación se necesita explotar la paralelización con mayor antelación, pudiendo aumentar o bien los recursos disponibles en reposo ($minVM = 5$) o bien el número de trabajos ejecutados por cada instancia ($N_{jobs} = 4$). Con la primera opción, se dispone desde el inicio de suficientes recursos con los que sacar provecho de la distribución de las simulaciones. De hecho, con 5 instancias iniciales se logra superar los tiempos alcanzados por las máquinas del laboratorio. El problema que acarrea este incremento es el aumento de los costes por el mantenimiento continuo de un mayor número de instancias en el estado de reposo.

Con la paralelización de trabajos dentro de cada instancia se consiguen resultados mucho más comedidos que con la solución anterior, pero a diferencia de ella supone

| | Número de simulaciones | | | | | |
|---|---|---|---|---|---|---|
| **Sistema** | **50** | **100** | **500** | **1.000** | **5.000** | **10.000** |
| Equipos del laboratorio | 88 | 177 | 889 | 1.809 | 9.444 | 17.330 |
| DNSE3, $minVM = 1, maxVM = 15, N_{jobs} = 1$ | 193 | 307 | 720 | 1.108 | 3.223 | 6.451 |
| DNSE3, $minVM = 1, maxVM = 30, N_{jobs} = 1$ | 193 | 307 | 720 | 1.108 | 2.848 | 6.322 |
| DNSE3, $minVM = 1, maxVM = 15, N_{jobs} = 4$ | 104 | 197 | 489 | 761 | 2.026 | 4.462 |
| DNSE3, $minVM = 1, maxVM = 30, N_{jobs} = 4$ | 104 | 197 | 489 | 761 | 1.971 | 3.317 |
| DNSE3, $minVM = 5, maxVM = 15, N_{jobs} = 1$ | 63 | 113 | 456 | 722 | 3.173 | 6.766 |
| DNSE3, $minVM = 5, maxVM = 30, N_{jobs} = 1$ | 63 | 113 | 456 | 722 | 2.717 | 3.913 |
| DNSE3, $minVM = 5, maxVM = 15, N_{jobs} = 4$ | 40 | 68 | 242 | 424 | 1.757 | 4.676 |
| DNSE3, $minVM = 5, maxVM = 30, N_{jobs} = 4$ | 40 | 68 | 242 | 424 | 1.518 | 2.808 |

Tabla 3.2: Tiempos de respuesta (en segundos) promedios para las ocho configuraciones diferentes del DNSE3 y los equipos del laboratorio utilizados como referencia

un coste cero ya que no se ha modificado en ningún momento la configuración de las instancias. Esta mejoría es bastante remarcable ya que, si se replicase el sistema en una máquina real con un único procesador, cuando un proceso requiriese de operaciones de entrada y salida, el planificador se encargaría de mantener el procesador ocupado con otro proceso. Sin embargo, en un entorno virtual, el hipervisor es quien decide qué CPU se asigna a cada una de las instancias, pudiendo asignar el procesador anfitrión a otra máquina virtual, por lo que el rendimiento experimentado depende de la carga a procesar de la máquina anfitriona más que de la virtual.

Mientras que para simulaciones cortas es más agresiva la paralelización con un mayor número de recursos mínimos, para simulaciones más largas se obtienen mejores resultados cuando se ejecutan más simulaciones en cada una de las instancias. A largo plazo, la configuración más conservadora y la que tiene un mayor número de recursos iniciales tienden a utilizar el mismo número de recursos y se nota en menor medida este despliegue inicial adicional, mientras que con la otra solución se procesa una mayor cantidad de simulaciones en paralelo.

Al basarse en mecanismos diferentes y completamente independientes, se ha probado la configuración que combine ambas mejoras ($minVM = 5, maxVM = 15, N_{jobs} = 4$). Para las simulaciones cortas, logra la reducción del tiempo por el mayor aprovisionamiento de los recursos pero ligeramente mejorada por la paralelización *hardware* en cada una de las instancias. En las simulaciones más largas saca partido de este mayor número de ejecuciones en paralelo para reducir los tiempos de simulación, siendo muy parecidos a la configuración con menor cantidad de instancias iniciales.

El último parámetro que se ha barrido en esta pruebas es el límite de instancias de simulación que pueden estar activas simultáneamente. La virtualización de la nube permite hacer un *overcommit* (es decir, ofrecer un número bastante más alto de CPUs virtuales que el número de CPUs físicas) gracias a que se presume una carga en las que las CPUs virtuales no están continuamente ocupadas. Sin embargo, los servicios de simulación con varias tareas asignadas aproximan la ocupación de la CPU al 100 %, por lo que no deja de haber recursos anfitriones suficientes para que el *overcommit* de la virtualización no se

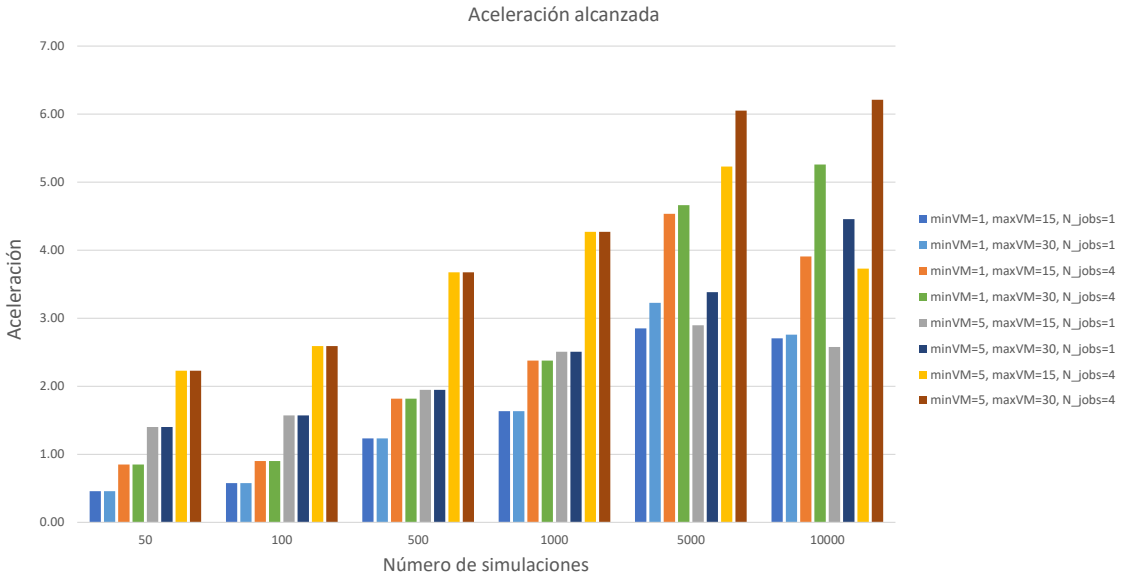


Figura 3.1: Aceleración alcanzada por cada una de las configuraciones del DNSE3 frente a los tiempos alcanzados en las máquinas de laboratorio para cada uno de los diferentes tamaños de las simulaciones.

perciba, y el rendimiento global se degrada.

En la figura 3.1, a modo de resumen, se muestran las aceleraciones de los tiempos logradas en las ocho configuraciones diferentes. Un dato relevante que se extrae de esta gráfica es la baja mejoría al pasar de 5000 a 10000 simulaciones, incluso en algunas simulaciones esta aceleración se ve reducida. Aunque este detrimento esta relacionado con el límite de las instancias de simulación, una vez se ha revisado la evolución temporal de las pruebas se ha detectado la presencia de ciertas deceleraciones aleatorias.

En las figuras 3.2a y 3.2b se muestra tanto la evolución temporal de las simulaciones pendientes de ejecutar (en azul) y la evolución de las instancias empleadas (en rojo) para las pruebas de 1000 y 10000 con la configuración más agresiva ($minVM = 5; maxVM = 30; N_{jobs} = 4$). Mientras que en la ejecución de las 1000 simulaciones se puede apreciar una aceleración progresiva en su procesado, en la prueba de 10000 aparecen ciertas deceleraciones mientras se continúa con la generación de nuevas instancias y una deceleración más apreciable cuanto más se aproxima al límite de las instancias, ilustrando la gran carga de trabajo a la que la infraestructura anfitriona se ve sometida. La aparición de estos eventos es aleatoria y no se ha podido predecir en la ejecución de estas pruebas, aunque se considera su relación con la temperatura de la máquina anfitriona. Se han revisado los registros del servicio de colas para detectar posibles bloqueos en la ejecución de alguna de las tareas, aunque no se han encontrado situaciones anómalas.

Los nodos de cómputo de la nube empleada en estas pruebas no se encuentran en las mejores condiciones de aclimatación, por lo que cuando se exige una gran carga de trabajo al sistema, supone un gran impacto en el rendimiento de la nube. Aunque no se dispone de acceso a los registros de los sensores de temperatura de estos nodos, se ha apreciado una reducción en la variabilidad de los resultados obtenidos cuando se incrementaba el tiempo de reposo entre las pruebas. En la figura 3.2c se puede apreciar cómo no se producen estos

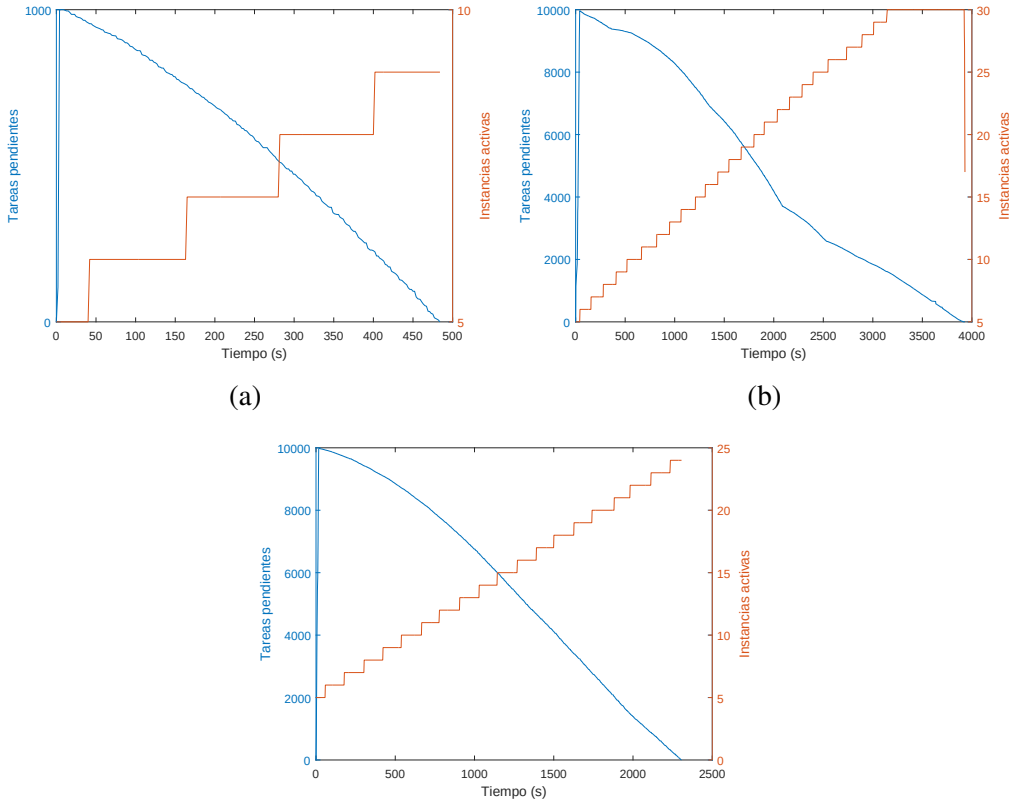


Figura 3.2: Evolución temporal de algunas de las pruebas realizadas para la medición del rendimiento del sistema, con la configuración $minVM = 5, maxVM = 30, N_{jobs} = 4$. En cada una de las figuras se muestra en azul el número de tareas pendientes de ejecutar en la cola durante la prueba y en rojo, el número de instancias activas. En (a) se muestra una de las pruebas de 1000 simulaciones individuales y en (b), una de las pruebas de 10000 simulaciones con claras pérdidas de rendimiento en su ejecución. En (c) se muestra otra de las pruebas de 10000 simulaciones en la que no se han producido esas deceleraciones.

eventos, logrando un tiempo de respuesta de 2306 segundos, 7,56 veces superior al tiempo medio obtenido en la ejecución local.

Estas figuras muestran también la baja velocidad con la que se lanzan nuevas instancias. El límite de instancias se alcanza cerca del término de la simulación, cuando deberían haberse lanzado con anterioridad al ser necesarias por la carga elevada de trabajo. En el futuro se deberá de analizar los beneficios aportados por un escalado más agresivo, como el ofrecido por las reglas propuestas en la sección 2.2.2.

Aun con estas limitaciones, especialmente el comportamiento tan conservador del escalado, se consigue alcanzar unos tiempos 6 veces menores que los obtenidos en las máquinas del laboratorio. En la revisión de los experimentos de la configuración más agresiva se ha calculado el numero medio de instancias utilizadas en las pruebas, alcanzando las 17 instancias en las simulaciones de 10000. Si revisamos los precios públicos de AWS, a día 17 de septiembre de 2017, los costes que supondría el despliegue del sistema se desglosarían en:

- 0,094 $ por hora en costes fijos por el uso de las instancias de los servicios de orquestación y de colas.
- 0,023 $ por hora en costes fijos por el uso de la instancia del servicio de informe.
- 0,115 $ por hora en costes fijos por las instancias de simulación activas en reposo.
- 0,305 $ en costes variables por el uso de las instancias de simulación durante la ejecución de las 10000 simulaciones

Aunque no se ha incluido el coste del servicio de almacenamiento, los costes que presenta son irrisorios en comparación a los presentes en la instalación y mantenimiento de los equipos de laboratorio para lograr resultado bastante superiores y recalca el beneficio ofrecido por el DNSE3. Con el acceso a recursos virtualmente infinitos en la nube pública, se podría plantear la utilización de reglas de escalado más agresivas que consiguiesen una aceleración incluso mayor a cambio de un mayor coste.

## 3.3. Usabilidad del sistema

Además de ofrecer una mejor respuesta temporal, el sistema debe ofrecer una buena experiencia de usuario que no suponga ninguna clase de impedimento en la ejecución de las simulaciones, especialmente en un entorno educativo. Esta prueba originalmente consistía en el uso del DNSE3 en el transcurso de las prácticas de las asignaturas de teletráfico. Los alumnos tendrían acceso al DNSE3 y a los equipos locales para la ejecución de las simulaciones y se les preguntaría por la experiencia percibida al término de cada práctica. Lamentablemente, cierto errores presentes en las librerías empleadas y otra serie de problemas adicionales impidieron su realización de forma exitosa

En su lugar, una vez finalizado los periodos lectivos de las asignaturas, se solicitó la ayuda de alumnos voluntarios que quisieran realizar una breve demo del DNSE3, siendo finalmente un total de 10. En esta demo se les pedía, de forma guiada, la realización de una breve réplica de la segunda práctica, que requiere de la ejecución de un mayor número de simulaciones. Posteriormente, el alumno debía realizar unas simulaciones adicionales, ahora sí de forma totalmente libre. Como el objetivo de la prueba era evaluar la experiencia de usuario más que el beneficio didáctico obtenido por el uso del sistema, solo se pidió a los voluntarios que revisaran que los experimentos se completaron con resultados razonables. Una vez se completaron estas pruebas, se encuestó a los voluntarios para reflejar sus percepciones de los dos sistema.

Aunque los voluntarios tuvieron acceso al DNSE3 durante un breve periodo de tiempo, los resultados obtenidos en las encuestas, según [Bro13], son similares a los obtenidos tras una exposición más larga, siempre y cuando esta se realice inmediatamente después de realizar la prueba.

Cada una de las encuestas realizadas presentaban una escala SUS (*System Usability Scale*) [Bro96], encontrándose traducida al español y con la versión modificada por [Ban08]. Esta escala ha sido ampliamente reconocida por ofrecer una estimación de la usabilidad de forma rápida pero consistente, y ha sido utilizada para evaluar *software* en diferentes entornos, incluyendo la docencia [Lin15, MG16]. consiste en 10 sentencias del

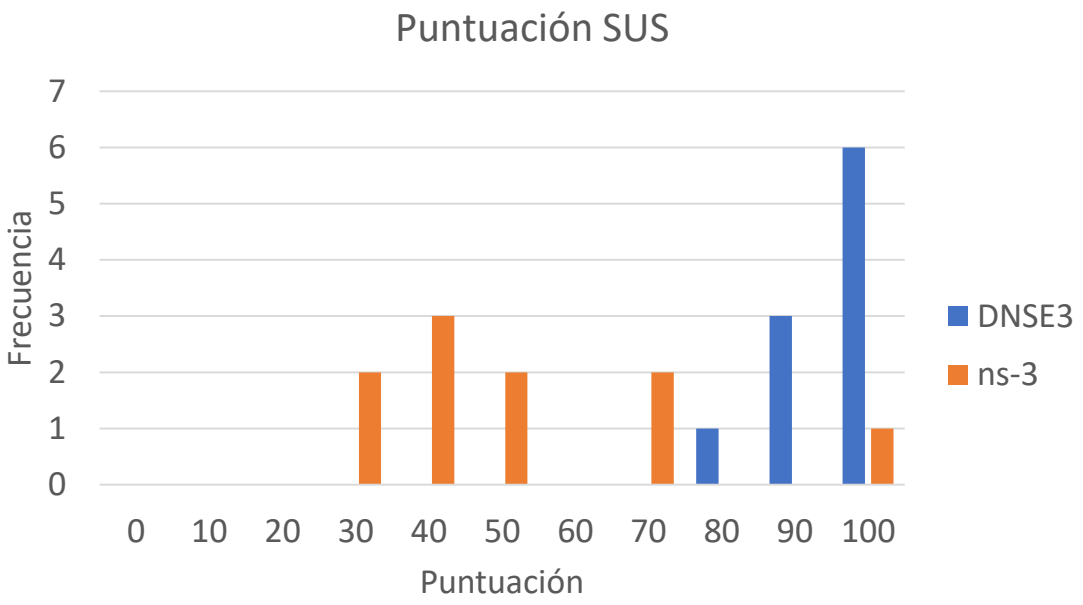


Figura 3.3: Distribución de las puntuaciones obtenidas en la encuesta SUS realizada por los voluntarios en ambos sistemas.

sistema evaluado, alternando su enfoque positivo y negativo, donde el encuestado debe reflejar su acuerdo o desacuerdo con un valor entre 1 y 5. Una vez agrupados todos los resultados se consigue un resultado comprendido entre 0 y 100. [Ban08] propone que no se reporte el resultado de cada una de las preguntas individuales, dada la alta correlación que presentan, a favor del resultado final. De hecho, presenta una regla de oro para traducir estos resultados en un adjetivo que lo describa.

Al final de cada encuesta, se han agregado tres preguntas adicionales, dos de las cuales les piden que presenten los aspectos positivos y negativos que encuentran en cada uno de los sistemas, mientras que la última les pregunta por su Probabilidad de Recomendación (*Likelihood to Recommend*, LTR) del sistema a otro usuario que presente la necesidad de usarlos, siendo en este escenario otro alumno que vaya a cursar la asignatura. El interés que suscita este último resultado es la fuerte correlación que presenta con el valor SUS, según han reportado [Lew14].

En la figura 3.3 se muestra la distribución de los resultados obtenidos de los 10 encuestados en ambos sistemas. De forma conjunta, el DNSE3 ha obtenido un resultado final de 91,75 (“excelente”) mientras el uso de los *scripts* logra alcanzar solamente 47,75 puntos (“pobre”). Dado el bajo número de encuestados no se pueden generalizar estos resultados, pero es bastante interesante que todos los encuestados otorgaron una mejor valoración al DNSE3, al igual que los resultados de cada una de las preguntas presentaban resultados más uniformes en el DNSE3 que en el uso directo del ns-3. Si revisamos el LTR, el DNSE3 obtuvo una puntuación de 5 frente a los 2,3 puntos de la otra herramienta, siendo consistente con el valor SUS presentado anteriormente.

En relación con los comentarios positivos y negativos que se solicitó a los encuestados, todos ellos mostraron su aprecio al uso de la interfaz web, que facilitaba la interacción

con el sistema. No por ello no mostraron algunos aspectos con los que no estaban de acuerdo, como la elección de las variables almacenadas en los ficheros de resultados o la necesidad de confirmar las diferentes acciones. Algunos de los voluntarios propusieron mejoras que se podían incluir en el sistema, como la presencia de un editor de texto que posibilitara la edición del modelo sin necesidad de emplear herramientas externas ni generar un nuevo proyecto a mayores. Algunos de ellos, curiosamente, reflejaron que detectaban una mejoría en los tiempos de ejecución.

# Capítulo 4

# Conclusiones

El uso de las simulaciones de barrido de parámetros ayuda a los estudiantes a comprender el comportamiento de los protocolos en las redes de comunicación y cómo se ven afectados por los diferentes parámetros presentes en ella. La principal complicación que presentan radica en el tiempo necesario para su ejecución por la gran cantidad de casos que deben procesar, pudiendo llegar a ser excesivo e inadmisible en estos entornos. Para reducir este tiempo, [SI16] presentó la aplicación DNSE3 que, aprovechando la paralelización de alto nivel ofrecida por las simulaciones individuales que conforman el barrido, presentaba un entorno donde procesar estas simulaciones de forma distribuida dentro de la nube computacional.

Entre las metas alcanzadas por este sistema se incluye el escalado automático de los servicios de simulación disponibles en función la demanda de trabajos solicitada, determinada por unas métricas generadas por la propia aplicación, y el uso de un mecanismo de asignación de tareas capaz de escalar simultáneamente con los servicios de simulación sin presentar pérdidas de rendimiento. Para lograrlo, este mecanismo relega la asignación a los servicios de simulación, según el modelo *work-stealing*, y evita mantener en todo momento el número de instancias disponibles.

Esta versión, aun siendo funcional, presentaba una serie de problemas y carencias, como la falta de persistencia de los datos de los usuarios, las colisiones en el acceso a las tareas de simulación y la falta de una política de escalado efectiva; que impedían su uso en un entorno multiusuario como son las sesiones de laboratorio de las asignaturas de teletráfico cursadas en los Grados en Ingeniería de Tecnologías y Tecnologías Específicas de la Telecomunicación de la Universidad de Valladolid. En este TFM se ha continuado con el trabajo realizado en el sistema y revisado su diseño con el propósito de ofrecer una versión final apta para el consumo del usuario final.

Entre los cambios directamente perceptibles por los usuarios del sistema, se ha incluido el respaldo de los datos de los usuarios entre sesiones de trabajo mediante el acceso a un gestor de bases de datos dentro del servicio de orquestación y se ha facilitado el acceso remoto al sistema gracias a un cliente gráfico web adaptado a los ordenadores personales y los dispositivos móviles y que puede utilizarse con las mismas credenciales empleadas en los laboratorios de las asignaturas.

La planificación de trabajos que disponía la versión original del DNSE3, a pesar de ser funcional, presentaba ciertos problemas como son la colisión de peticiones de los servicios de simulación, que intentan acceder a la misma tarea, o el acaparamiento de

los recursos por parte de alguno de los usuarios cuando solicitaban un elevado número de simulaciones. Para superar estas limitaciones se ha incorporado a la cola una rotación *round-robin* de los trabajos en dos niveles: de usuario y de trabajo. Para dos peticiones consecutivas preguntando por la siguiente tarea a procesar, esta rotación asegura que los dos trabajos no correspondan ni a la misma tarea ni al mismo usuario, siempre que haya elementos suficientes dentro de ella.

En lo que respecta al escalado, la aplicación original no ofrecía unas reglas de escalado reales más allá de las orientadas a testeo de la función. Es por ello que se ha diseñado una nueva política de escalado, consciente tanto del número de trabajos pendientes de procesar como del número actual de recursos disponibles, que haga un uso eficiente de las instancias de simulación, evitando fluctuaciones bruscas en sus creaciones y eliminaciones y manteniéndolas activas más tiempo del necesario para agilizar la ejecución de las tareas de simulación

Con el sistema final ya en funcionamiento, se ha realizado una evaluación del sistema, tanto en rendimiento como en usabilidad, contrastándolo con el uso del ns-3 mediante *scripts* de línea de comandos, según se ha utilizado en las prácticas de laboratorio. Los resultados obtenidos en las pruebas de rendimiento muestran una mejoría significante en los tiempos obtenidos para barridos de parámetros extensos incluso en la configuración que no aprovecha el paralelismo de forma agresiva, es decir, disponiendo de una única máquina activa en su estado de reposo y procesando una única simulación de forma simultánea en cada una de las instancias. Si el DNSE3 se configura para explotar la paralelización de las simulaciones más agresivamente, ya sea incrementando el número de recursos en reposo o aumentando el número de simulaciones simultáneas por instancia, se consigue mejorar estos resultados, incluso en simulaciones de tamaño más reducido. El uso conjunto de estos mecanismos logra reducir este tiempo hasta 6 veces lo percibido en el uso local del ns-3.

Aparte del rendimiento, se ha analizado la usabilidad del sistema gracias a la colaboración de una serie de estudiantes que han replicado las tareas de una de las prácticas de laboratorio realizada con los *scripts* del ns-3 esta vez con el cliente web. En las encuestas que realizaron mostraron una clara preferencia por la nueva aplicación, obteniendo altas calificaciones tanto la *System Usability Scale* y la Probabilidad de Recomendación del sistema.

El interés y actualidad que presenta este sistema se ha visto reflejado en la redacción de dos artículos presentando su propuesta y funcionamiento, de los cuales uno ha sido aceptado en las Actas de las XIII Jornadas de la Ingeniería Telemática y el otro está pendiente de revisión.

Adicionalmente, aunque dispone de cierto margen de mejora, el DNSE3 ofrece suficiente madurez como para utilizarse dentro de la realización de las prácticas de laboratorio de las asignaturas de teletráfico donde, además de reducir el tiempo de ejecución de las simulaciones, permitiría evaluar los beneficios pedagógicos que ofrece o estudiar los patrones de peticiones de simulación de los usuarios con el fin de proponer nuevas políticas de escalado, esta vez adaptadas a la tendencia de los usuarios.

## 4.1. Líneas de trabajo futuro

A pesar de los prometedores resultados obtenidos al término de las diferentes pruebas, el sistema todavía presenta un margen con el que mejorar aún más su respuesta. Estas propuestas, de nuevo, se centran en la asignación de los trabajos y de las políticas de escalado.

Con la inclusión de los dos niveles de rotación dentro de la cola de tareas, se ha conseguido evitar tanto el acaparamiento de los recursos por parte de los usuarios y las colisiones entre dos o más servicios de simulación intentando acceder al mismo trabajo. Este mecanismo, según se indicó en la sección 2.2.1 puede incluir un tercer nivel de rotación que permita la ejecución de tareas de simulación del mismo usuario que procedan de diferentes simulaciones. Aunque se utiliza actualmente la rotación *round-robin*, que ofrece un reparto equitativo de los recursos, se puede estudiar la inclusión de pesos que establezca una preferencia a la ejecución de determinadas tareas.

Donde se dispone de un mayor abanico de posibilidades de mejorar es en el escalado. Con el uso de otras métricas más allá del volumen de tareas publicada pueden garantizar una mejor estimación de la carga de trabajo que debe procesar. Por otro lado, el crecimiento de las reglas presentadas resulta muy conservador y retrasa el despliegue de los recursos requeridos. Con la propuesta de crecimientos más agresivos, como puede ser la duplicación de los recursos disponibles, se espera mejorar los tiempos de respuesta de las simulaciones al disponer de estos recursos con mayor antelación.

Además de proponer alternativas a las políticas de escalado, sería conveniente definir mecanismos de evaluación que tuviesen en cuenta tanto el tiempo de respuesta alcanzado (o la aceleración con respecto de una configuración base) como el coste de los recursos consumidos, bien medido de una manera genérica (p.ej. $\frac{MV}{horausadas}$), bien expresado en términos económicos según las tarifas de un eventual proveedor de nube pública (como se ha hecho en el capítulo 3). De esta manera, podría preferirse la política de escalado que balancease adecuadamente ambos aspectos.

En lo que respecta al cliente web, aun con las buenas críticas recibidas, los usuarios han mostrado la necesidad de disponer de herramientas integradas dentro del sistema que faciliten la edición de los proyectos sin tener que crear uno nuevo. Esta característica podría ser incluida dentro del servicio de orquestación mediante la incorporación de un sistema de versionado que identifique los parámetros y resultados asociados a cada una de esta versiones. Además, con el fin de reducir el número de herramientas externas necesarias para las simulaciones, se puede llegar a integrar el servicio de estadística, eliminado en esta versión, que ofrezca a los alumnos la posibilidad de operar sobre los resultados de las simulaciones y reportarlos en los informes de las prácticas.

# Bibliografía


[Adr93] W Richards Adrion. Research methodology in software engineering. En *Summary of the Dagstuhl Workshop on Future Directions in Software Engineering" Ed. Tichy, Habermann, and Prechelt, ACM Software Engineering Notes, SIGSoft*, volumen 18, páginas 36–37, 1993. 5

[Arm10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 2010. 3

[Ban08] Aaron Bangor, Philip T. Kortum, and James T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human & Computer Interaction*, 24(6):574–594, 2008. 28, 29

[BL04] Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, and Eduardo Gómez-Sánchez. Grid characteristics and uses: A grid definition. En Francisco Fernández Rivera, Marian Bubak, Andrés Gómez Tato, and Ramón Doallo, editores, *Grid Computing: First European Across Grids Conference, Santiago de Compostela, Spain, February 13-14, 2004. Revised Papers*, páginas 291–298, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. 2

[BL12] Miguel L. Bote-Lorenzo, Juan I. Asensio-Pérez, Eduardo Gómez-Sánchez, Guillermo Vega-Gorgojo, and Carlos Alario-Hoyos. A grid service-based distributed network simulation environment for computer networks education. *Computer Applications in Engineering Education*, 20(4):654–665, 2012. 2, 3

[Blu99] Robert D. Blumofe and Charles E. Leiserson. Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720–748, Septiembre 1999. 14, 15

[Bro96] John Brooke. SUS: A quick and dirty usability scale. En Patrick W. Jordan, B. Thomas, Ian Lyall McClelland, and Bernard Weerdmeester, editores, *Usability evaluation in industry*, páginas 189–194. Taylor & Francis, London, UK, June 1996. 28

[Bro13] John Brooke. Sus: A retrospective. *J. Usability Studies*, 8(2):29–40, Febrero 2013. 28

[Buy09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and

reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009. 3

[CP12] Rafael Cano Parra. Entorno de simulació de redes TCP/IP usando servicios REST basados en la nube computacional. Poyecto Fin de carrera, Universidad de Valladolid, Valladolid, España, 2012. 3, 4

[Fie14] R. Ed. Fielding and Ed. J. Reschke. Hypertext transfer protocol (http/1.1): Conditional requests. RFC 7232, RFC Editor, June 2014. 12

[Fos01] Ian Foster. The globus toolkit for grid computing. En *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid*, página 2, 01 2001. 2

[Law91] Averill M. Law and W. David Kelton. *Simulation modeling and analysis*. McGraw-Hill, New York, NY, USA, 1991. 1

[Lew14] James R. Lewis. Usability: Lessons learned... and yet to be learned. *International Journal of Human & Computer Interaction*, 30(9):663–684, 2014. 29

[Lin15] Hao-Chiang Koong Lin, Mei-Chi Chen, and Chih-Kai Chang. Assessing the effectiveness of learning solid geometry by using an augmented reality-assisted learning system. *Interactive Learning Environments*, 23(6):799–810, 2015. 28

[MG16] Anabel Martin-Gonzalez, Angel Chi-Poot, and Victor Uc-Cetina. Usability evaluation of an augmented reality system for teaching euclidean vectors. *Innovations in Education and Teaching International*, 53(6):627–636, 2016. 28

[Pap02] C. Papadopoulos and J. Heidemann. Using ns in the classroom and lab. En *ACM SIGCOMM Workshop on Computer Networking*, páginas 45–46, Pittsburgh, PA, USA, 2002. 1

[Qun08] Zhao A. Qun and Wang Jun. Application of ns2 in education of computer networks. En *IEEE International Conference on Advanced Computer Theory and Engineering*, páginas 368–372, 2008. 1

[Ric08] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008. 8

[SI16] Sergio Serrano Iglesias. Entorno de simulació de redes distribuido basado en ns-3 y computación en nube. Poyecto Fin de carrera, Universidad de Valladolid, Valladolid, España, 2016. 3, 4, 5, 7, 9, 31

[Vaq08] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, Diciembre 2008. 3

[Vaq13] Luis Miguel Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Cloud scalability: building the millennium falcon. *Concurrency and Computation: Practice and Experience*, 25(12):1623–1627, 2013. 3

[Vin07] S. Vinoski. Rest eye for the soa guy, 2007. 8

[Wan12] S. Y. Wang, C. C. Lin, Y. S. Tzeng, W. G. Huang, and T. W. Ho. Exploiting event-level parallelism for parallel network simulation on multicore systems. *IEEE Transactions on Parallel and Distributed Systems*, 23(4):659–667, April 2012. 1, 2

[Wei09] E. Weingartner, H. vom Lehn, and K. Wehrle. A performance comparison of recent network simulators. En *IEEE International Conference on Communications*, páginas 1–5, June 2009. 1

[Zha10] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, May 2010. 3

# Apéndice A

# Artículo presentado en las Actas de las XIII Jornadas de la Ingeniería Telemática

Este artículo fue redactado en abril de 2017, donde se reflejan algunas de las innovaciones presentadas en este TFM: la rotación *round-robin* de dos niveles de las tareas de simulación, la persistencia de los datos de usuario, la definición de las reglas de escalado y la disponibilidad de una interfaz de usuario.

La compilación única de los modelos y la evaluación de la usabilidad del sistema no fueron reflejados al realizarse con posterioridad a su redacción. Por otra parte, la nube utilizada en las pruebas disponía únicamente del primero de los nodos indicados en el capítulo 3 (dos procesadores Intel(R) Xeon(TM) E5-2603 de 6 núcleos a 1,60 GHz).

JITEL 2017
XIII Jornadas de Ingeniería Telemática
27-29 Septiembre 2017, Valencia (España)

*Actas de las XIII Jornadas de Ingeniería Telemática (JITEL 2017), Valencia (España), 27-29 de Septiembre de 2017.*

ISBN: X-XXXX-XX-XXX-X
DOI: XXXXXX

# Entorno de simulación distribuida de redes basado en la nube computacional

Sergio Serrano Iglesias, Eduardo Gómez Sánchez, Miguel L. Bote Lorenzo,
Juan I. Asensio Pérez, Manuel Rodríguez Cayetano
Departamento de Teoría de la Señal, Comunicaciones e Ingeniería Telemática
Universidad de Valladolid
ETSI de Telecomunicación, Paseo de Belén 15, 47011 Valladolid
sergio@gsic.uva.es, edugom@tel.uva.es, migbot@tel.uva.es, juaase@tel.uva.es, manrod@tel.uva.es

***Resumen*—Las simulaciones de barridos de parámetros tienen un gran potencial en el estudio de redes telemáticas, especialmente en contextos docentes. Sin embargo, el elevado tiempo necesario habitualmente para completar este tipo de simulaciones es una limitación importante para su uso. En este artículo se propone DNSE3, un entorno que permite la ejecución distribuida de tareas de simulación en el simulador ns-3 dentro de un entorno de nube computacional, a través de una arquitectura de servicios RESTful. El sistema se ha diseñado para ser autoescalable, aprovisionando y liberando dinámicamente recursos de la nube computacional en función de la carga de simulaciones demandada, y garantizando un reparto equitativo de los recursos entre los distintos usuarios. Además, DNSE3 se ha implementado reutilizando servicios presentes en *middlewares* de nube populares, y ha sido evaluado mediante pruebas sintéticas. La implementación de DNSE3 ha demostrando un correcto comportamiento funcional y un rendimiento considerablemente superior a otras alternativas cuando el número de simulaciones es muy elevado.**

***Palabras Clave*—Nube computacional, simulación, barrido de parámetros, escalado, entorno distribuido, REST**

## I. INTRODUCCIÓN

La simulación es una herramienta de gran ayuda para el estudio de redes telemáticas. Ofrece un mecanismo de evaluación de protocolos y despliegues que no son fácilmente caracterizables mediante modelos analíticos. Del mismo modo, permite estudiar el comportamiento de las redes sin tener que recurrir a instalaciones reales, generalmente costosas [1], [2]. Entre los usos que se dan a estas simulaciones se encuentran la investigación y desarrollo de nuevos estándares y protocolos, así como la caracterización del comportamiento del despliegue de una infraestructura ante la llegada de diferentes patrones de tráfico [2]. Algunos de los simuladores de redes más empleados y conocidos son el ns-2 [3], muy utilizado en investigación; el ns-3 [4], siguiente versión del ns-2; OMNet++; [5] y Riverbed Modeller [6].

La simulación de redes es también una aproximación muy utilizada en entornos académicos. Gracias a las simulaciones, los alumnos pueden reforzar los conocimientos adquiridos en las sesiones teóricas mediante la replicación de los escenarios planteados. Las simulaciones, a su vez, permiten estudiar redes que no se pueden poner fácilmente en marcha con equipamiento real (por falta de recursos o potenciales problemas de seguridad); o, incluso, estudiar redes que sí están disponibles, pero en situaciones que son difíciles de reproducir en la realidad.

Para el análisis de redes es muy habitual recurrir a las denominadas simulaciones de barrido de parámetros, en las cuales algunos de sus parámetros toman valores dentro de un rango definido por el usuario para ver cómo se ve afectado el modelo de simulación. Entre los usos que se dan a este tipo de simulaciones se incluye la revisión del desempeño de la red en función de los valores de los distintos factores que entran en juego en la comunicación (velocidades binarias de enlaces, tamaños de ventana, etc.) o la optimización de dichos factores para alcanzar determinados requisitos deseables de calidad de servicio.

Normalmente la ejecución de un barrido de parámetros implica la realización de múltiples simulaciones individuales, tantas como número de combinaciones diferentes de valores de parámetros sean posibles dentro de los rangos definidos por el usuario. En un ordenador convencional, estas simulaciones se ejecutarán secuencialmente. Aunque el tiempo de ejecución de una única simulación puede ser relativamente corto, en una de barrido de parámetros éste se ve incrementado por el gran volumen de procesos necesarios para completarla, lo que alarga la espera hasta disponer de los resultados finales. Dentro del contexto educativo, este retardo puede afectar negativamente al desarrollo de las prácticas de laboratorio, que disponen de un tiempo limitado en sus sesiones de trabajo.

Dado que cada una de las simulaciones en las que se descompone un barrido de parámetros constituye una tarea individual e independiente del resto, se puede re-

ducir el tiempo requerido para su finalización mediante la ejecución de varias simulaciones en paralelo, ya sea explotando las capacidades de computación paralela de un ordenador (múltiples núcleos) o de un sistema distribuido (múltiples ordenadores) [7]. En la computación paralela en un ordenador se reparten los diferentes trabajos en hilos y procesos de ejecución diferentes, con el fin de sacar el máximo rendimiento de los recursos de los que dispone la máquina. Algunos de los simuladores de redes, como el ns-3 u OMNet++, incorporan esta técnica para la ejecución de las simulaciones. Aunque se pueden conseguir mejoras en el rendimiento, la escalabilidad de esta solución se ve limitada por el número de núcleos o procesadores instalados en el único ordenador donde se realiza la simulación [7].

Por su parte, en la computación distribuida se utilizan múltiples ordenadores para repartir los diferentes trabajos a realizar. Así, la escalabilidad no está limitada por los recursos de computación de un ordenador, ya que pueden incorporarse ordenadores adicionales. Por el contrario, los límites a la mejora del rendimiento los define la capacidad de la red y la necesidad de sincronizar los diferentes procesos, en el caso de que haya dependencias [7]. Como se ha mencionado, las simulaciones resultantes de un barrido son totalmente independientes, no existiendo este coste de sincronización. Por ello, si una simulación se completa en un tiempo $T$, $N$ simulaciones podrían completarse en $N$ ordenadores en un tiempo marginalmente superior a $T$. Sin embargo, y hasta donde sabemos, los simuladores de redes telemáticas existentes no hacen uso de esta técnica.

Dada esta problemática, los autores de este artículo desarrollaron el DNSE (*Distributed Network Simulation Environment*) [8], un entorno que permitía la ejecución de simulaciones de barrido del simulador ns-2 dentro de una infraestructura de *grid* computacional. Este entorno tuvo una buena acogida entre los profesores y alumnos de la E.T.S. de Ingenieros de Telecomunicación de la Universidad de Valladolid. A pesar del buen funcionamiento que mostraba, el sistema presentaba una serie de problemas ligados a la propia infraestructura de *grid* en la que funcionaba. El escalado del sistema se tenía que administrar manualmente, lo que propiciaba situaciones en las que o bien el entorno disponía de recursos aprovisionados sin utilizarse, o bien no se alcanzaban tiempos de respuesta suficientemente bajos que mejorasen la calidad de servicio. Otro problema importante era la considerable dificultad técnica y administrativa que suponía la puesta en marcha de un *grid* en el que participaran múltiples organizaciones administrativas con el objetivo de asegurar la cantidad de recursos necesarios para soportar el uso del DNSE a gran escala. Un problema adicional que existía en la época en la que se estuvo utilizando fue la falta de madurez que ofrecían los *middlewares* disponibles para el desarrollo de aplicaciones orientadas al grid.

En los últimos años ha surgido un paradigma que consigue solucionar algunos de estos problemas: la nube computacional. La nube computacional ha cobrado gran popularidad con la oferta de nubes públicas, entre las que se encuentran Google App Engine [9], Microsoft Azure [10] o Amazon Web Services (AWS) [11]. Las denominadas nubes de infraestructura como servicio (*Infrastructure as a Service*)[12] presentan una serie de características que cubren las carencias del *grid*, como son: el aprovisionamiento de recursos bajo demanda y su gestión remota, lo que permite un despliegue rápido de máquinas virtuales y simplifica su administración; y la monitorización de los servicios y recursos empleados y el escalado rápido de éstos [12]. Estas dos últimas características, junto con la capacidad de ofrecer recursos aparentemente infinitos [12], permiten realizar un seguimiento del sistema y ajustar el número de máquinas desplegadas a las necesidades computacionales. Además, gracias al interés comercial generado por este paradigma [12], el *middleware* disponible es mucho más estable y robusto, existiendo varias alternativas, tanto en la nube pública como para desplegar nubes privadas, con interfaces de servicios muchas veces compatibles.

En este artículo se presenta el DNSE3 (*Distributed Network Simulation Environment 3*), un rediseño importante inspirado en el sistema anterior, que introduce cambios significativos: por un lado, cambia la infraestructura del *grid* por la nube computacional gracias a las ventajas que aporta; por otro, aprovecha la evolución del simulador ns-2 a ns-3, con sus correspondientes mejoras de rendimiento [2]; finalmente, introduce funcionalidades detectadas como necesarias en el estudio anterior.

A lo largo de este artículo se presentarán los siguientes apartados. En primer lugar se revisarán los trabajos que es posible encontrar en la literatura con propuestas relacionadas con la realización de simulaciones de forma distribuida. A continuación, se tratará el diseño del DNSE3, discutiendo los principales requisitos y la propuesta de arquitectura resultante. La siguiente sección estará enfocada en los principales problemas que debe resolver el DNSE3 y la forma en la que se han tratado. Finalmente se discutirá sobre las conclusiones y el trabajo futuro.

## II. ESTADO DEL ARTE

En la literatura se pueden encontrar diferentes proyectos que han permitido el uso distribuido de los simuladores de diferentes campos de estudio. Uno de ellos es el DNSE [8], que, según se mencionó en la sección anterior, distribuía las simulaciones de barrido de parámetros de redes telemáticas para el simulador ns-2 dentro de un *grid* computacional. Aparte de mejorar el tiempo de respuesta de las simulaciones, ofrecía una GUI (*Graphic User Interface*) para facilitar tanto la gestión de los trabajos de simulación como el visionado de las animaciones generadas a partir de la herramienta NetAnim, ofrecida de forma conjunta con el ns-2.

Otro ejemplo de despliegue en el *grid* fue DSoG (*Distributed Simulation on Grid*) [13], que facilita el estudio y modelado de motores de aviación. Esta propuesta, a diferencia de las mencionadas en esta sección, no pretende optimizar la ejecución de las simulaciones. En su lugar,

ofrece un entorno colaborativo en el que sus usuarios puedan compartir sus datos y modelos, de tal forma que se dispone de una amplia biblioteca con multitud de recursos con los que generar los modelos de simulación finales.

Si nos centramos ahora en aquellos proyectos que hagan uso de la nube, se pueden encontrar dos entornos orientados a la simulación de tráfico automovilístico: SEMSim CS (*Scalable Electro-Mobility Simulation Cloud Service*) [14], usado principalmente para el estudio del escenario en el que se reemplazasen todos los vehículos actualmente existentes por una alternativa eléctrica, y $C^2$SuMo (*Cloud-based, Collaborative ans Scale-up Modelling and Simulation Framework for STEM Education*) [15], un entorno educativo para estudiantes de educación superior con el que estudiar simulaciones ambientadas en el mundo real. En ambas propuestas se aprovecha la flexibilidad de la nube para mejorar el tiempo de respuesta de las simulaciones.

Aunque existen otros simuladores que hacen uso de la nube computacional, no existen propuestas que empleen la nube computacional enfocadas al estudio de las redes telemáticas y, los que sí están enfocados, utilizan otro tipo de infraestructuras. Con la propuesta de este artículo se espera cubrir esta carencia.

## III. DISEÑO DEL DNSE3

En esta sección se cubre el diseño del DNSE3. En primer lugar se presentan los requisitos más importantes que debe cumplir para utilizarse en un ambiente multiusuario con acceso concurrente, para, finalmente, presentar la arquitectura empleada.

### A. *Requisitos del DNSE3*

El DNSE3 es un entorno de simulación distribuida de redes para ser usado en contextos académicos. Debe satisfacer un conjunto de funcionalidades básicas para que los alumnos gestionen simulaciones de manera sencilla, y debe funcionar de manera robusta y eficiente, para ser un ayuda y no un impedimento al aprendizaje. A continuación se detallan brevemente los requisitos impuestos en el diseño del DNSE3.

En cuanto a los **requisitos funcionales**, el sistema debe:

- **Gestionar los proyectos de simulación**. Los usuarios deben ser capaces de crear proyectos de simulación, en los que se definen el modelo a emplear en las diferentes simulaciones, los parámetros soportados por el modelo y los resultados que se generan tras su ejecución. Toda esta información es necesaria para que el sistema pueda ejecutar adecuadamente las simulaciones y se puedan recuperar los resultados deseados.
- **Ejecutar simulaciones de barrido de parámetros**. Además de la ejecución de simulaciones individuales, se debe permitir a los usuarios la libre configuración de los parámetros y la selección de los resultados para el barrido.
- **Controlar la ejecución de las simulaciones**. El usuario podrá elegir, en todo momento, si quiere iniciar la ejecución de una simulación previamente creada o detenerla, en caso de que quiera modificar alguno de sus parámetros o ficheros recolectados. Los usuarios pueden confundirse en los valores proporcionados y deben ser capaces de alterar dichos valores sin tener que esperar a que se termine la ejecución completa y descartar los resultados generados. Por otro lado, en caso de detectar un fallo en la ejecución de las simulaciones, se deberá notificar al usuario, detener la ejecución de todas las tareas asociadas y esperar a que el usuario lo resuelva antes de continuar con su ejecución.
- **Recuperar los resultados**. Tras la ejecución de las simulaciones, los ficheros de resultados se deben almacenar de forma persistente, incluso después de que el usuario haya recibido una copia. En el caso de los resultados de los barridos, el sistema debe aclarar al usuario qué resultado se corresponde con qué combinación de parámetros.

Por otra parte, existen una serie de **requisitos no funcionales** que condicionan el diseño y las decisiones tecnológicas acerca del DNSE3 y que se enumerarán a continuación.

- **Escalabilidad del sistema**. El tiempo requerido para la ejecución de las simulaciones de barrido no deberá crecer de forma proporcional al número de simulaciones derivadas de estas. Por este motivo, será necesario utilizar múltiples máquinas virtuales que ejecutarán las simulaciones en paralelo. Para conseguir optimizar tanto los tiempos de respuesta como los recursos utilizados, el sistema debe ser autoescalable. Para ello, deberá contar con una cantidad de recursos suficiente y ser capaz de determinar el número de recursos necesarios para las simulaciones solicitadas, ya sea aprovisionado o liberándolos, con el menor impacto posible en el resto de funciones.
- **Reparto de recursos**. El sistema será empleado por diferentes alumnos, y se debe garantizar un reparto equitativo de los recursos disponibles entre todos ellos. Será necesario incorporar mecanismos que impidan que un alumno (de manera intencionada o no) acapare todos los recursos del sistema e impida el progreso de las tareas del resto de usuarios.
- **Tolerancia a fallos**. El sistema debe reaccionar a los fallos en la infraestructura (se pierde alguna máquina virtual) o de alguno de sus servicios (se bloquea un proceso de simulación) ocultando estos eventos al usuario final y replanificando las tareas, de manera que en todo caso la degradación sea paulatina (*graceful degradation*).
- **Clientes de acceso**. Se debe disponer de clientes, tanto en línea de comandos (orientados a pruebas o usuarios avanzados, y de los que no se hablará en este artículo) como a través de una interfaz web, que permitan el control remoto de las operaciones del sistema.

### *B. Arquitectura del sistema*

El DNSE3 presenta una arquitectura orientada a servicios (SOA, *Service Oriented Architecture*) que propone la separación de las diferentes funciones de un sistema en diferentes servicios con una interfaz de acceso expuesta a cualquier clase de cliente. Con este tipo de arquitectura obtendremos un sistema más sencillo de construir, mantener y escalar [16]. Para las aplicaciones desplegadas en una nube de computación, este tipo de arquitectura ofrece ventajas en su administración. Una vez configurados los servicios que funcionarán en cada una de las instancias, se puede generar una instantánea que mantenga su configuración en ese momento y, cuando se produzca un fallo en alguna de las máquinas, se puede crear una nueva máquina a partir de esa instantánea. De igual manera, cuando sea necesario replicar servicios para conseguir escalar hacia arriba, sólo será necesario levantar nuevas máquinas virtuales a partir de estas imágenes.

Cada uno de los servicios que forman parte de la aplicación DNSE3 ha sido diseñado siguiendo el estilo arquitectural REST (*REpresentational State Transfer*), que propugna una arquitectura orientada a recursos (ROA, *Resource Oriented Architecture*) y un conjunto de interfaces homogéneo. De las diferentes características de REST, como son el uso de una interfaz uniforme o la navegación a través de sus recursos, la que más nos interesa ahora es la carencia de estado de peticiones previas o *statelessness*. Los servicios RESTful, según este principio, no mantendrán información relacionada con ninguna petición previa de cualquiera de sus clientes, sino que estos deberán mantener esa información. Esto facilita el equilibrado de carga, al poder dirigir las peticiones a diferentes réplicas de un servicio y, por ello, la tolerancia a fallos y la escalabilidad.

La propuesta final de la arquitectura del DNSE3 se compone de un total de siete servicios, mostrados en la figura 1, entre los que nos podemos encontrar el *servicio de orquestación*, encargado de recibir las peticiones de los usuarios y coordinar al resto del sistema mediante el envío de tareas de simulación al servicio de; el *servicio de colas*, que gestiona el reparto de tareas de simulación; el *servicio de simulación*, capaz de ejecutar las simulaciones de los usuarios; el *servicio de informe*, que se encarga de preparar los resultados de las simulaciones para que puedan ser utilizados posteriormente por los usuarios; el *servicio de almacenamiento*, que ofrece un almacenamiento persistente compartido por el resto de servicios; el *servicio de monitorización*, que recolecta métricas que muestran el uso que se hace del sistema, y el *servicio de escalado*, capaz de ajustar el número de instancias de simulación a la demanda de trabajos de los usuarios.

La propuesta final de la arquitectura del DNSE3 se compone de un total de siete servicios, mostrados en la figura 1, entre los que nos podemos encontrar el *servicio de orquestación*, encargado de recibir las peticiones de los usuarios y coordinar al resto del sistema mediante el envío de tareas de simulación al servicio de; el *servicio de colas*, que gestiona el reparto de tareas de simulación; el *servicio*

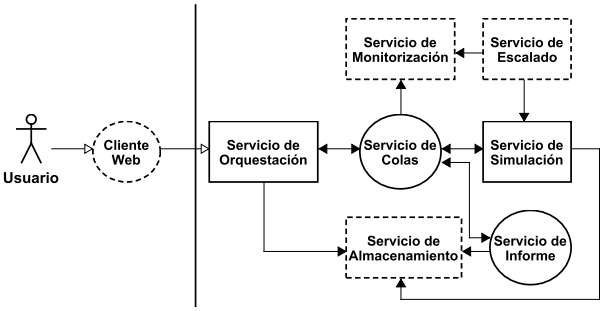


Figura 1. Servicios que componen el DNSE3 y las interacciones que establecen.

*de simulación*, capaz de ejecutar las simulaciones de los usuarios; el *servicio de informe*, que se encarga de preparar los resultados de las simulaciones para que puedan ser utilizados posteriormente por los usuarios; el *servicio de almacenamiento*, que ofrece un almacenamiento persistente compartido por el resto de servicios; el *servicio de monitorización*, que recolecta métricas que muestran el uso que se hace del sistema, y el *servicio de escalado*. Este último servicio es de vital importancia para el correcto desempeño de la aplicación, al ser el encargado de ajustar el número de instancias de simulación a la demanda de trabajos de los usuarios mediante la definición de una serie de reglas basadas en las métricas del servicio de monitorización.

Cada uno de los servicios previamente mencionados presentan diferentes grados de acoplamiento con la aplicación, lo que facilita su agrupación basada en la facilidad que presentan para ser integrados en otros proyectos diferentes. De esta forma, los servicios de orquestación y simulación presentan las funciones más ligadas a la aplicación (el primero gestiona el flujo de esta aplicación, el segundo envuelve e invoca al ns-3) y ofrecen mayor dificultad para ser utilizado en otros ámbitos. Por otro lado, los servicios de colas e informe presentan funciones más genéricas y fácilmente extensibles (podrían servir para, con facilidad, desplegar otro sistema paralelo y escalable y en otras infraestructuras diferentes de la nube). Finalmente, es interesante observar que los servicios de almacenamiento, monitorización y escalado son de uso habitual en diversas aplicaciones desplegadas en esta clase de entornos y son ofrecidos de forma nativa por muchos *middlewares* de nube, por lo que, a diferencia de los anteriormente mencionados, no necesitan ser desarrollados *ad-hoc* sino que se accederá a su API pública. Esta distinción de los servicios se ve reflejada en la figura 1, perteneciendo los servicios representados con un rectángulo con trazo sólido, un círculo con trazo sólido y un rectángulo con trazo discontinuo, respectivamente, a cada una de las tres agrupaciones mencionadas.

## IV. IMPLEMENTACIÓN DE LOS SERVICIOS

### A. *Aspectos tecnológicos*

Los entornos de nube computacional presentan una gran heterogeneidad. Por una parte, se presentan diferentes modelos de nube en función de la abstracción que nos ofrezcan de su infraestuctura, como son IaaS, PaaS (*Platform as a Service*) o CaaS (*Container as a Service*), entre otros. Por otra parte, tanto las instancias desplegadas como las máquinas anfitrionas no están sujetas a un tipo de arquitectura o sistema operativo concreto. De entre los diferentes modelos de nube, se ha optado por un diseño orientado a IaaS para el DNSE3, al ofrecer un mayor nivel de gestión y configuración requeridos por algunos servicios, como sucede con el de simulación.

El despliegue se ha realizado en una nube privada que utilizaba OpenStack [18] como sistema gestor de la infraestructura. Este *middleware*, muy utilizado en nubes privadas, ofrece un entorno IaaS compatible con AWS, con el que se ofrece la posibilidad de extender a una nube híbrida, orientada a cubrir picos de demanda cuando los recursos de la nube privada no sean suficientes. Entre los servicios de OpenStack destacan Swift y Cinder, para el almacenamiento distribuido basado en objetos y volúmenes, respectivamente; Ceilometer, para la publicación de métricas; y Heat que, junto a Ceilometer, permiten gestionar las políticas de escalado del sistema. Todos estos servicios han sido integrados dentro del DNSE3 por medio de la invocación a su API REST.

Dada la libre configuración de entornos en la nube, se ha decidido desarrollar los diferentes servicios del DNSE3 en Java, al tratarse de un lenguaje multiplataforma muy popular que facilita la libre elección del entorno empleado. Además, existen diferentes *APIs* (*Application Programming Interface*) para el despliegue de servicios RESTful en Java. En el DNSE3 se ha empleado la API Restlet [17], que permite el desarrollo tanto de servidores RESTful como clientes REST, está disponible para diferentes plataformas (Java SE, Java EE, OSGI...) y ofrece un gran abanico de funcionalidades adicionales como son las implementaciones de seguridad para la integración de otros servicios.

Otro aspecto importante en una arquitectura SOA es la representación de la información intercambiada en las interfaces de servicio. Para reducir el volumen del tráfico cursado, se utilizarán documentos JSON (*JavaScript Object Notation*) para el intercambio de mensajes. Este formato de documento es mucho más liviano que otro tipo de lenguajes de marcado, como sucede en XML (*eXtensible Markup Language*). Esta simplicidad y ligereza conlleva la pérdida de la validación de la estructura del documento (que sí está disponible en XML). Este problema no supone un impacto mayor gracias a la disposición de librerías como GSON o Jackson, que permiten la conversión de los datos contenidos en el documento a objetos de clases conocidas o desarrolladas.

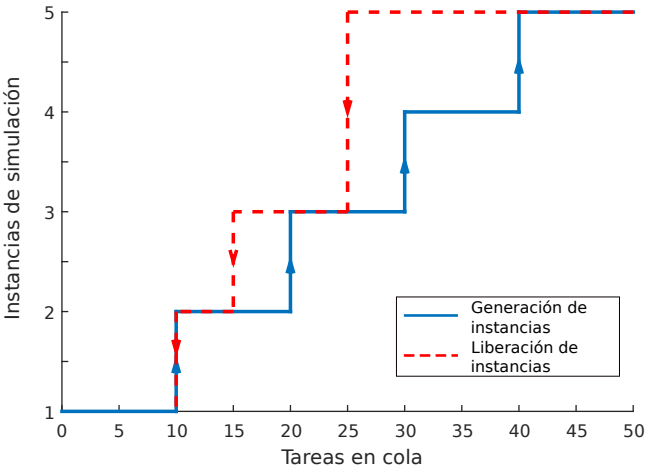


Figura 2. Representación de cómo las reglas de escalado definidas van haciendo aumentar el número de instancias disponibles según aumentan los trabajos pendientes en la cola de simulación (línea azul, progreso de izquierda a derecha), y de cómo esas mismas reglas van eliminando instancias al reducirse el número de trabajos pendientes (línea roja, progreso de derecha a izquierda).

### B. *Escalado del sistema*

Esta función es la más importante del DNSE3 y la que aporta mayor valor añadido respecto a la versión anterior. El sistema debe ser capaz de acomodar el número de recursos desplegados a la demanda de trabajos de simulación solicitada por los usuarios. Para que este ajuste sea lo más eficiente posible, se debe lograr, por un lado, evitar continuas fluctuaciones en el número de instancias utilizadas y, por otro, mantener la mayor ocupación posible.

El tiempo de arranque de las máquinas no es despreciable y podría suceder que, una vez se haya lanzado una nueva máquina, se decida eliminar otra de las desplegadas al haberse reducido la demanda en ese tiempo de carga. Esta situación supondría un desperdicio de recursos de la máquina anfitriona, con un impacto negativo en el rendimiento.

Así, en lugar de eliminar las instancias tan pronto como se reduzca la carga, interesa que las instancias sólo sean eliminadas cuando la carga se reduzca considerablemente. Este sobreaprovisionamiento temporal permitirá mejorar el tiempo de respuesta y reaccionar rápidamente a nuevas subidas de carga y, en cualquier caso, desaparecerá cuando el número de tareas descienda de manera importante.

La política de escalado empleada en el DNSE3 para determinar cuándo y cómo se efectúa este escalado y que sea capaz de cumplir los requisitos anteriores se basa en el número de trabajos que debe efectuar cada una de las instancias de simulación, tal y como se ilustra en la figura 2. Hasta que esta proporción no supera un determinado umbral (en nuestro caso 10 simulaciones por instancia), no se aprovisionarán nuevos recursos de simulación. En el caso contrario, mientras no se reduzca la carga de cada instancia al 50 % del umbral anterior (5 simulaciones por instancia), se mantendrá el número de recursos desplegados. Una vez se reduzca, se liberan la

mitad de las instancias desplegadas.

Una ventaja de esta política es su sencillez de implementación, Los servicios de escalado de los entornos de nube permiten incluir políticas basadas en reglas *IF-THEN*, en las que se comparan los valores almacenados en el servicio de monitorización con un umbral previamente fijado, o bien la agrupación de éstas para generar reglas más complejas. En nuestro caso sólo se necesitan 2 reglas de comparación: si $\frac{tareas}{instancias} > 10$, entonces $instancias++$; y, si $\frac{tareas}{instancias} < 5$, entonces $instancias = 0{,}5 \times instancias$, redondeando al entero mayor. En las reglas anteriores el término $tarea$ agrupa tanto a las simulaciones pendientes de ejecutar como las que están siendo procesadas por las diferentes instancias de simulación. Para que estas métricas puedan aplicarse, el servicio de colas deberá enviar a Ceilometer una métrica que contenga el reparto equitativo de las tareas solicitadas (dato conocido al contener todo el listado de trabajos) entre las instancias de simulación (que deberá de solicitar de forma periódica a Heat).

Cuando la carga de trabajo decrece a los niveles indicados por la política de escalado, se procede a la eliminación de las instancias de simulación sobrantes. En el caso de que estas máquinas estén ejecutando alguna simulación, se intentará esperar a que ésta se complete sin solicitar un nuevo trabajo antes de eliminarla, aunque puede darse la situación de que se interrumpa dicho trabajo. En este último escenario, los trabajos asociados volverían a estar disponibles una vez haya expirado su tiempo de reserva al no recibir actualizaciones del servicio de simulación.

### C. *Asignación de trabajos de simulación*

Un aspecto fuertemente ligado al escalado es el lugar en el que se realiza la planificación de recursos. Si el servicio de colas actuase de *scheduler* centralizado, debería conocer qué servicios de simulación están disponibles, cuáles se están apagando, si se están levantando nuevos, y sus URI de acceso. Como se puede prever, esto supone un aumento de complejidad en el servicio.

En su lugar, se ha seguido un modelo de asignación *work-stealling*, en el que, en lugar de asignar los trabajos a los diferentes trabajadores, son estos los que solicitan el trabajo que deben realizar. Cuando se inicia una nueva instancia, ésta conoce de antemano la dirección de acceso al servicio de colas para poder preguntar por el siguiente trabajo a realizar. El servicio de simulación está siempre intentando trabajar, así que o está ocupado, o demanda nuevos trabajos al servicio de colas. Además, periódicamente informa al servicio de colas de su actividad, de manera que cuando deja de comunicarse con él, el servicio de colas puede asumir que ha caído. Como en los servicios RESTful no se mantiene información de las conexiones previas, este diseño permite que el servicio de colas sólo se tenga que preocupar de mantener la cola de trabajos y determinar el trabajo siguiente a procesar, tareas que no dependen del número de instancias de simulación que haya, lo que simplifica enormemente la escalabilidad.

En la determinación de la siguiente tarea a ejecutar se realiza una rotación (*round-robin*) de dos niveles entre los trabajos pendientes, ilustrada en la figura 3, con el fin de evitar tanto el acaparamiento del sistema por parte de un usuario (de forma intencionada o no) como la recolección de un mismo trabajo por parte de dos instancias diferentes. El primer nivel de rotación va alternando de usuario propietario de los trabajos, de forma que dos trabajos solicitados consecutivos no pueden pertenecer a un mismo usuario (salvo que no haya trabajos pendientes de otro usuario), mientras que el segundo nivel recorre los trabajos pendientes de ejecutar del usuario escogido en el nivel anterior. Dentro de los trabajos pendientes se incluyen aquellos no asignados y los asignados anteriormente a un servicio que ha caído (que ha dejado de reportar su actividad).

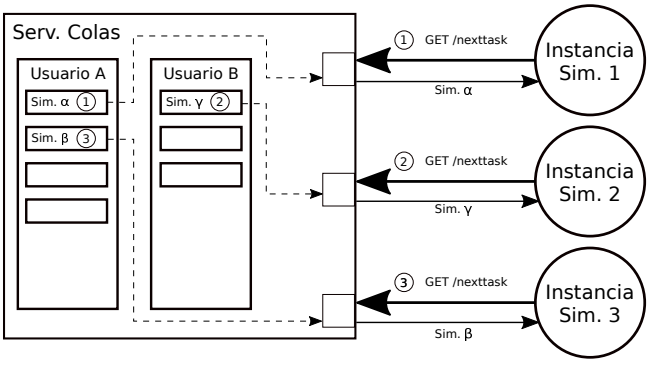


Figura 3. Ilustración de la asignación del trabajo siguiente mediante la rotación *round-robin* de los usuarios propietarios de los trabajos.

## V. INTERFAZ DE USUARIO

Aunque los servicios RESTful, gracias a su interfaz uniforme, que en el caso de los servicios web se corresponde con los métodos de del protocolo HTTP (*Hyper-Text Transfer Protocol*), permiten usar cualquier cliente web (por ejemplo, un navegador) para ser invocados, es complejo que el usuario final prepare y consuma los cuerpos de peticiones y respuestas. Es por ello que se ha optado por desarrollar un cliente del DNSE3 que es una aplicación web con una interfaz de usuario ilustrada en la figura 4. Este cliente se encarga de recibir y procesar las peticiones de los usuarios finales, adecuándolas a la interfaz REST del servicio de orquestación y de reproducir las respuestas recibidas en representaciones HTML visualmente agradables para devolver al navegador. Para que esta aplicación sea usable desde cualquier navegador web y tipo de dispositivo, se ha desarrollado usando el *framework* Bootstrap [19], que ofrece soporte con los dispositivos móviles como *smartphones* o tabletas, muy populares entre los alumnos en los últimos años.

Adicionalmente, la seguridad se puede gestionar de manera delegada: el servicio de orquestación acepta todas las peticiones provenientes del cliente web, mientras que éste es el encargado de validar las credenciales de los usuarios. Como no se aceptan en el servicio de orquestación peticiones de otros orígenes, resulta más sencillo proteger al DNSE3 de ataques externos.

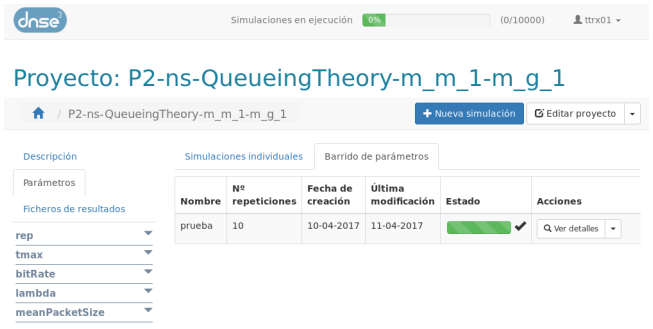


Figura 4. Intrefaz web del DNSE3

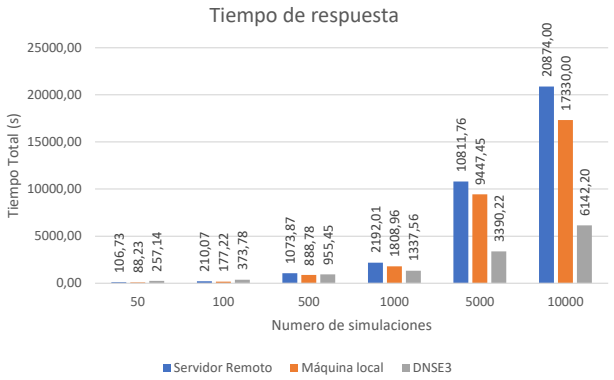


Figura 5. Tiempos de respuesta ofrecidos por cada uno de los sistemas evaluados ante diferentes tamaños de simulaciones a ejecutar

## VI. EVALUACIÓN DEL RENDIMIENTO

Una vez se ha completado el desarrollo de una versión de pruebas del sistema, se ha evaluado si podría ser desplegada en un entorno de producción. Las pruebas realizadas incluyen el acceso concurrente de varios usuarios, que no debe comprometer la calidad de servicio (QoS, *Quality of Service*) del resto de usuarios; la publicación de simulaciones; la recolección de los resultados; y el reparto de los recursos según se comentó en la sección C.

A falta de una evaluación con usuarios reales, en lo que respecta a temas de rendimiento, se han efectuado una serie de pruebas sintéticas para evaluar el escalado automático de las instancias de simulación. Estas pruebas han consistido en la ejecución de un barrido de parámetros sintético consistente en varias simulaciones de duración más o menos semejante. Estas pruebas se han repetido en tres sistemas: el servidor del laboratorio de la asignatura (ordenador con procesador Intel(R) Xeon(R) E5-2620 de 4 núcleos a 2 GHz y 8GB de memoria RAM), que los alumnos podrían usar fuera del horario de prácticas; los ordenadores del laboratorio (ordenadores con procesador Intel(R) Core(TM) i5-2400 de 4 núcleos a 3,10 GHz y 4 GB de memoria RAM), sólo accesibles en horario lectivo; y el DNSE3. En los dos primeros casos se han buscado momentos en los que no había otros usuarios corriendo procesos en el sistema. En el caso del DNSE3 se partía de una situación de reposo (un único servicio de simulación creado) en cada experimento, y el servicio de escalado podría llegar a crear hasta un máximo de 15 instancias simultáneas de servicios de simulación.

La figura 5 muestra los tiempos requeridos para la ejecución completa de todas las simulaciones, para barridos de parámetros sintéticos consistentes en un distinto número de trabajos. El tiempo típico para completar una simulación individual ha rondado los 2-3 segundos tanto en las máquinas locales como en el servidor del laboratorio, mientras que en el DNSE3 este tiempo se incrementaba hasta los 7 segundos. Este incremento se debe a las comunicaciones realizadas entre los diferentes servicios: la publicación de la tarea en el servicio de colas, la recogida de la tarea por parte del servicio de simulación, la obtención del modelo de simulación almacenado en el servicio de almacenamiento y la posterior carga de los resultados y la notificación del estado de ejecución al servicio de orquestación.

En vista a los resultados obtenidos, se pueden distinguir dos situaciones diferentes. En primer lugar, si el número de simulaciones a ejecutar es bajo, en barridos que den lugar hasta unas 500 simulaciones, el DNSE3 no presenta una ventaja comparativa: por una parte, el número de simulaciones es bastante bajo y puede completarse secuencialmente en un tiempo pequeño. Aunque el servicio de escalado aprovisiona nuevas instancias, estas tardan en arrancar unos 75 segundos, por lo que llegan a asumir pocas tareas, obteniéndose un beneficio muy limitado de la paralelización. Por otra parte, a medida que aumenta el número de simulaciones totales a ejecutar, las nuevas instancias levantadas por el servicio de escalado llegan con todavía muchos servicios pendientes de procesar en la cola, así que en un momento dado se están realizando un gran número de ejecuciones en paralelo, compensando sobradamente la sobrecarga de la distribución, llegándose a alcanzar una reducción del 65 % del tiempo de respuesta con barridos de 5000 simulaciones.

Aún así, como en el entorno de pruebas se ha restringido el número de máximo de servicios de simulación, a partir de un punto el tiempo de respuesta se incrementa de forma lineal. Para evitar esta situación, se puede hacer uso de una nube híbrida y así conseguir recursos virtualmente infinitos.

## VII. CONCLUSIONES Y TRABAJO FUTURO

El uso de los simuladores de redes telemáticas es beneficioso para la investigación y análisis de despliegues de infraestructura y protocolos de comunicaciones. Aun así, las simulaciones de barrido de parámetros tienen un coste computacional excesivo, lo que limita su utilización especialmente en entornos educativos, con recursos computaciones y tiempo escasos.

Para tratar este problema, este artículo ha propuesto el DNSE3, una aplicación distribuida orientada a servicios que, aprovechando las ventajas de la nube computacional, introduce mecanismos de autoescalado para aprovisionar recursos dinámicamente en función del número de simulaciones demandadas por los alumnos. Su diseño también ha tenido en cuenta otros aspectos relevantes, como el reparto

equitativo de los recursos entre usuarios y la tolerancia a fallos.

Para ilustrar su funcionamiento, se ha programado un prototipo de la aplicación con una interfaz de usuario web. Posteriormente, se ha evaluado el rendimiento del escalado sometiendo al DNSE3 a barridos de simulación sintéticos de distintos tamaños, observándose que cuando el número de simulaciones individuales es bastante elevado el tiempo de respuesta obtenido por el DNSE3 es hasta un 65 % menor que el alcanzado con las otras alternativas disponibles en la ETSIT de Valladolid. Además, se ha observado que el comportamiento era funcionalmente correcto, y que se lanzaban y detenían servicios de simulación en función del número de tareas en la cola, siguiendo las reglas de escalado definidas.

Aun así, las pruebas realizadas hasta ahora se han hecho en situaciones controladas por los autores de este trabajo. Es, por lo tanto, necesario repetir este estudio con usuarios reales (alumnos) en un contexto real (muchos alumnos compitiendo simultáneamente por los recursos). Este nuevo estudio permitirá validar los resultados obtenidos anteriormente y, además, recoger el patrón de peticiones de simulación llevado a cabo por los usuarios reales, lo que a su vez facilitará la propuesta de nuevos experimentos sintéticos más realistas en el futuro. También se aprovecharán estas pruebas para conocer la opinión de los usuarios acerca de aspectos de usabilidad e interfaz de usuario, así como para indagar en el impacto que el uso de esta aplicación tiene en sus procesos de aprendizaje.

Otra vía de trabajo futuro se refiere a la exploración de alternativas a las políticas de escalado descritas en la sección B, para estudiar el coste/beneficio de utilizar políticas más agresivas o el mantenimiento de un número mínimo de instancias de simulación permanentemente disponibles.

## AGRADECIMIENTOS

Este trabajo ha sido parcialmente financiado por los proyecto TIN2014-53199-C3-2-R del Ministerio de Economía y Competitividad, y VA082U16 de la Junta de Castilla y León, con cofinanciación FEDER. Los autores agradecen al resto del grupo de investigación GSIC/EMIC por su apoyo e interés en el proyecto.

## REFERENCIAS

[1] A. M. Law and W. D. Kelton, *Simulation modeling and analysis*. McGraw-Hill, 1991.

[2] E. Weingartner, H. vom Lehn, and K. Wehrle, "A Performance Comparison of Recent Network Simulators," in *2009 IEEE International Conference on Communications*, June 2009, pp. 1–5.

[3] "The network simulator - ns-2," 2017. [Online]. Available: http://www.isi.edu/nsnam/ns/

[4] "The network simulator - ns-3," 2015. [Online]. Available: https://www.nsnam.org/

[5] "Omnet++ discrete event simulator," 2017. [Online]. Available: https://omnetpp.org/

[6] *Riverbed Modeler*, Riverbed Technology, 2015. [Online]. Available: https://www.riverbed.com/es/products/steelcentral/steelcentral-riverbed-modeler.html

[7] R. M. Fujimoto, "Research Challenges in Parallel and Distributed Simulation," *ACM Transactions on Modeling and Computer Simulation*, 2016.

[8] M. L. Bote-Lorenzo, J. I. Asensio-Pérez, E. Gómez-Sánchez, G. Vega-Gorgojo, and C. Alario-Hoyos, "A grid service-based Distributed Network Simulation Environment for computer networks education," *Computer Applications in Engineering Education*, 2010.

[9] *Google App Engine*, Alphabet Inc., 2017. [Online]. Available: https://cloud.google.com/appengine/?hl=es

[10] *Microsoft Azure*, Microsoft Corporation, 2017. [Online]. Available: https://azure.microsoft.com/es-es/

[11] *Amazon Web Servicies*, Amazon Inc., 2017. [Online]. Available: Available: https://aws.amazon.com/es/

[12] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of Cloud Computing," *Communications of the ACM*, 2010.

[13] Y. Cao, X. Jin, and Z. Li, "A distributed simulation system and its application," *Simulation Modelling Practice and Theory*, 2007.

[14] D. Zehe, A. Knoll, W. Cai, and H. Aydt, "SEMSim Cloud Service: Large-scale urban systems simulation in the cloud," *Simulation Modelling Practice and Theory*, 2015.

[15] F. Caglar, S. Shekhar, A. Gokhale, S. Basu, T. Rafi, J. Kinnebrew, and G. Biswas, "Cloud-hosted simulation-as-a-service for high school STEM education," *Simulation Modelling Practice and Theory*, 2015.

[16] S. Vinoski, "REST Eye for the SOA Guy," *IEEE Internet Computing*, 2007.

[17] *Restlet Framework*, Restlet, Inc., 2017. [Online]. Available: https://restlet.com/open-source/

[18] *OpenStack*, OpenStack Foundation, 2017. [Online]. Available: https://www.openstack.org/

[19] "Bootstrap," 2017. [Online]. Available: http://getbootstrap.com/

# Apéndice B

# Artículo pendiente de revisión

Este borrador artículo data de finales de junio de 2017. En este artículo se han incluido las decisiones de diseño no presentadas en el artículo anterior. Adicionalmente, se han realizado un mayor número de experimentos con el nuevo hardware, con nuevas configuraciones y un mayor tamaño de las muestras de las simulaciones más largas para reducir la varianza de los resultados. Estos resultados no están todavía incorporados en el borrador del artículo mostrado a continuación

# A self-scalable distributed network simulation environment based on cloud computing

S. Serrano-Iglesias, E. Gómez-Sánchez, M.L. Bote-Lorenzo, J.I. Asensio-Pérez, M. Rodríguez-Cayetano

*Department of Signal Theory, Communications and Telematics Engineering*
*School of Telecommunications Engineering, Universidad de Valladolid*
*Paseo de Belén 15, 47011 Valladolid, Spain*

**Abstract**

Parameter sweep simulations have a great potential in understanding computer networks, but they convey a significant computational load. This paper proposes DNSE3, a service oriented application that makes use of cloud computing to exploit parallelization in order to achieve reduced response times for parameter sweeps. It features a decentralized scheduling mechanism that, along with application level workload metrics and user defined scalability rules, achieves self-scalability, by provisioning and freeing resources for simulation as needed. When evaluated with large simulation parameter sweeps, DNSE3 achieves significant gains in performance. Usability surveys in an educational context also show a relevant user acceptance.

*Keywords:* distributed simulation, computer networks simulation, automatic scalability, cloud computing

## 1. Introduction

Computer network simulation is often used by researchers or standard proponents and reviewers, since it allows to evaluate protocols or network configurations that cannot be deployed in reality with ease or within a reasonable

*Email addresses:* sergio@gsic.uva.es (S. Serrano-Iglesias), edugom@tel.uva.es (E. Gómez-Sánchez), migbot@tel.uva.es (M.L. Bote-Lorenzo), juaase@tel.uva.es (J.I. Asensio-Pérez), manrod@tel.uva.es (M. Rodríguez-Cayetano)

budget [1, 2], and to explore extreme traffic situations which are rare but of critical impact [2]. The importance of simulation in this domain has fostered the production of open source and proprietary simulators, such as ns-2 [3], often used in research, its successor ns-3 [4], OMNet++ [5], and Riverbed Modeler [6].

Besides in research and the industry, network simulation is much used in academic context, since it helps students easily understand the behavior of a protocol under diverse network configurations, without the cost of a real infrastructure and the security concerns entailed by letting students manipulate them. For example, [7] reports using ns-2 both in the classroom, where the teacher illustrates some concepts by running simulations and visualizations, and in the laboratory, where students propose and run simulated scenarios in order to evaluate the impact of protocol design choices. Similarly, [8] wraps ns-2 into a more friendly system that can be used by the teacher as a "demonstrator", while students set up experiments with a graphical interface, run them and analyze the results. More recently, [9] employ ns-3 so that students can observe both the dynamic behavior and the performance metrics of a wireless local network. Nevertheless, in these studies teachers propose one or just a small set of network configurations for the students to simulate. Pedagogically, it would be more interesting to let students freely explore the parameter space in order to search for relevant phenomena and understand their causes. For example, [10] describes a learning scenario in which learners must understand the relationship between one output variable (TCP throughput) and some network parameters (link delays and bit rates). Since the output depends on both inputs, this experiment demands a parameter sweep simulation that can be explored as a $2^k r$ factorial design [11, chapter 18]. In a typical situation in which students use standard desktop computers available in a learning laboratory the response times would be very high. This fact renders the learning scenario unfeasible in practice, since it is not advisable that students spend most of their laboratory time waiting for the simulation to complete.

In spite of the high computational load of a parameter sweep study, it is im-

portant to notice that it can be decomposed into a (typically large) number of *independent* simulations, each of them running the same model with a different set of parameters. Thus, hardware solutions that can exploit thread or process level parallelism can aid to bring down simulation times [12]. However, in order to achieve sensible reductions that permit carrying out parameter sweeps in a laboratory session with many students, a significant amount of hardware is needed. Investing an important budget in such equipment is not advisable, especially if it only serves one course or type of courses, as this hardware shall most likely remain unused out of laboratory teaching periods. Instead, an infrastructure that may scale up and down easily would fit much better to short intermittent peaks of computation needs, as is the case of parameter sweep simulations in education.

In a previous work, we developed DNSE [10], a simulation environment especially designed to carry out educational parameter sweep simulators based on ns-2. This application made use of a service oriented computational grid infrastructure [13], where simulation services could be deployed and released in the computation nodes so that simulation times could be shortened when needed. Despite its good performance in peak situations, and its utility for learning attested by students [10], DNSE had some relevant limitations. On one hand, the system was scalable but manual intervention of the administrator was needed to add up or remove computation nodes. With a dynamic load, this implies that at certain times the system may be overprovisioned, while some others resources are not enough to attain fair response times. On the other hand, while the philosophy of creating virtual organizations ("federations of resources" spanning multiple administrative domains) is appealing, the experiments in [10] were carried out in a grid belonging to a single organization, and even nowadays it is not straightforward to set up a service oriented grid (in particular one based on the Web Services Resource Framework, WSRF [14]) due to its reduced adoption (since the publication of the standard, in 2006, the OASIS consortium has not published updated or events). This facts challenges the potential for both arbitrary fast scalability and reduction of ownership costs.

Instead, the cloud computing paradigm claims several merits that can ease the development of a parameter sweep network simulation environment. Building on virtualization technologies, it allows on demand provisioning under a pay-per-use business model, without up front investment [15]. Thus, a system can be easily dimensioned according to the estimated resource needs. Moreover, reserving and releasing resources can be made quickly without human intervention, through user defined scalability rules that consider both infrastructure and application specific metrics [16]. This paves the way for flexible, virtually infinite scalability of an application at reasonable costs [17]. Finally, the popularity of cloud computing has led to the consolidation of some prominent public cloud platforms, such as Google App Engine [18], Microsoft Azurre [19] or Amazon Web Services (AWS) [20], but also to the emergence of robust middleware to set up private clouds. Interestingly, some of the latter like OpenStack [21] or Cloudstack [22] offer service interfaces and packaging formats compatible with AWS, easing the realization of hybrid clouds. An application running in such environment would make use of locally owned resources in stationary situations, while peak demands could be overflowed to the public infrastructure at a small cost.

This paper presents DNSE3 (*Distributed Network Simulation Environment 3*), an application conceived for education that allows students to define, run and analyze both simple simulations and parameter sweep simulations. While its service architecture is inspired in that of the DNSE [10], it includes many innovations: it is based on ns-3 [4], profiting from some of its improvements over ns-2 (e.g. better integration with widespread data analysis and visualization tools or the possibility of embedding real network stack code in simulations [4], and much better performance in the simulation of wireless networks [23]); it makes use of several standardized cloud services to optimize performance and reliability; it redefines totally the scheduling approach, to better suit a more distributed environment in which the amount of resources could vary quickly; and most importantly, it takes on the scalability features of cloud computing, by defining application level metrics and automatic scalability rules so that the

cloud middleware will recruit or release resources as needed to achieve short response times that make it feasible to support learning while in a laboratory session. It should also be noticed that though the user interface is thought mostly attending to its use as learning tool, there are no other design decisions influenced by this fact, and the application can be used for research or the industry, as long as the simulation models of ns-3 are found convenient.

The rest of this paper is structured as follows. Section 2 reviews other distributed simulators found in the literature. Next, section 3 describes the main decisions behind the service oriented design of DNSE3, including a detailed discussion on how to achieve automatic scalability, while section 4 briefly presents some technological choices taken while developing a fully functional prototype. The results of evaluating DNSE3 are reported in section 5, considering both performance and usability issues. Finally, conclusions are drawn in the last section.

## 2. Related work

Parallel and distributed simulation has been a field of research for quite some time [24]. In many cases, a single simulation model is reprogrammed so that distinct computational resources evaluate some of the simulation events, being thus synchronization the most critical issue [24]. A somewhat different approach of parallelization delivers separate but dependent simulation models to each of the computing elements (the *logical processes*, LP). For instance, [12] refactors ns-2 to distribute a large network topology among several simulation threads, so that each one simulates completely some routers and links. However, as these topology parts exchange traffic between them, the LPs in charge of them have to wait for events coming from other LPs. Similarly, [25] proposes dSUMO, a decentralized software architecture to simulate urban traffic in which each LP is in charge of a partition of a city model (consisting of interacting elements such as cars or pedestrians). For evaluation, they deployed this architecture in a virtualized environment (virtual machines and networks) dimensioned be-

forehand, though their research focused on the impact of different partitioning mechanisms, not on scalability.

In all the above proposals, tasks split among LPs are, at some point, dependent of each other. In contrast, the aforementioned DNSE [10] pursued performing parameter sweep simulations benefiting from the fact that each of them is a completely independent process. It was designed to distribute the workload in a service oriented grid. A broker service found out from a directory which simulation services are available and split the parameter sweep between them, collecting the results when simulations were finished. Noticeably, the simulation services did not communicate among them. [26] also uses the computational grid to support simulation, by proposing a framework to develop simulations that can be discovered, instantiated and used in the grid, illustrated with the simulation of an aircraft engine model. The simulation themselves do not benefit from parallelization, though each different client may ask a factory to create a dedicated instance of the simulation service.

The cloud computing paradigm has also been adopted to conduct high performance simulation. SEMSim [27] uses it to run the back-end of an urban traffic simulator. With the front-end tools, the user designs an experiment that submits as a "simulation bundle" to a dispatch server in the back-end. This dispatcher is in charge of provisioning a number of virtual machines to run independent simulations in parallel. This number, however, is determined beforehand. [28] claims to follow a similar approach, though the models they simulate are not restricted to a particular domain.

Finally, cloud computing has also been used to support education in many ways [29], among them supporting simulation. The *Cloud-based collaborative and Scale-up Modelling and Simuation Framework for STEM Education* ($C^2$SuMo) has been designed with some educational constrains in mind, like supporting collaboration and being very easy to learn, and is aimed at letting students construct, refine and simulate urban traffic models. A virtual machine running a SUMO simulator [30] is launched for each user, though the work demanded by an individual user is not actually parallelized.

## 3. DNSE3 design

The DNSE3 is an application originally aimed at carrying out simulations of computer networks for educational purposes, including parameter sweeps. For the shake of usability, it should offer simple yet significant functionalities to manage simulations and their results. In addition, non functional requirements such as scalability are key driving forces behind this proposal. This section first discusses these requirements, then proposes a service oriented architecture distributing the functional responsibilities, and finally discusses in detail some keys to achieve self-scalability: a distributed scheduling approach, the publication of application level workload metrics, and the evaluation of scalability rules.

### *3.1. Functional requirements*

The proposed application should allow users to **upload, view and remove simulation projects**. A simulation project includes a C++ simulation model suitable for ns-3 (i.e. that can be compiled with the ns-3 libraries), consisting of a network topology and default values for the parameters, indicating which of them can be later modified by the student though the DNSE3 graphical interface. Note that, unlike in [7], where a few sets of simulation problems are embedded in the application, this decision allows to benefit from ns-3 flexibility to simulate arbitrary network topologies. In particular, in the educational context the teacher may distribute distinct simulation projects for each assignment or different courses, without restriction. It should be noted that when stating which parameters may be editable, a simulation project may have none, thus being ready to be run after uploading it.

Once uploaded to the platform, the user should be able to **configure simulation projects**. The span of decisions may vary from setting a *single value* to a parameter different than the default, specifying *a range of values* for it (hence requesting a parameter sweep), requesting the repetition of the same simulation several times (in order to study the effect of some random variable) or providing details on how DNSE3 should handle the output of the simulations (e.g.

storing all of them, including simulation traces, just some performance metrics, computing some statistics...).

After projects are ready, users can **run single simulations, parameter sweeps, or multiple repetitions of either of them**. While they are running, the system should allow users to **monitor the progress of a simulation, pausing and resuming it as needed**. Also, the system should inform the user of problems arising when running a simulation.

Finally, the system should **store permanently the results** of the simulations carried out until the user removes them. In the case of a parameter sweep, the system will report the results of each possible combination of parameters.

### *3.2. Non functional requirements*

The main motivation for the proposal of DNSE3 is to achieve **scalability** in the computation of parameter sweep simulations. Therefore, when one or several parameter sweeps are requested, the system should provision enough resources to compute simulations in parallel in order to complete them within a reasonable response time. On the contrary, when just a few simulations are demanded, idle resources should be released. Therefore, the system should be able to *automatically* provision and free computational resources as the workload varies.

Some additional issues should be considered in the design. First, the system should make a **fair allocation of resources**, so that no single user may fetch most of them while the rest have to wait until the scalability mechanisms provision some more. Moreover, as in any distributed system, some parts may fail. The system should have some degree of **fault tolerance**, hiding these problems from the user, even if they eventually cause some performance degradation. Finally, the system architecture should decouple clearly business logic from user interface, so that multiple interfaces can be offered, some aimed at fast learnability and usability, especially in the educational domain, while others based on calling an API or using a command line may yield better productivity in research or industrial contexts.

*3.3. System architecture*

The DNSE3 architecture has been designed following the principles of Service Oriented Architectures (SOA) [31], separating the functionalities in several services with clearly defined interfaces. This approach simplifies maintenance and offers a way to scalability by replicating the most demanded services and balancing the computational load between them [32]. One simple way to do this in a cloud computing platform that operates *Infrastructure as a Service* (IaaS) is to host each of the services in a virtual machine. The application thus scales up by launching or shutting down virtual machines that run the required services as needed to meet the existing load. Robustness can also be attained by duplicating critical services in different virtual machines.

In addition, the design of each of the services complies to the *REpresentational State Transfer* (REST) architectural style [33], that advocates services should offer an interface inspired by the Resource Oriented Architecture (ROA), i.e. exposing a series of resources and a very limited and standardized methods to manipulate them, instead of a wide range of full fledged methods, as this strategy has allowed a plethora of third party browsers and web servers interact successfully in the World Wide Web [33]. REST also promotes that services should be stateless, i.e. each request should include all necessary information to be processed on itself and should be treated independently of previous requests. This particular feature is key to load balancing and thus fault tolerance and scalability [31].

Following these guidelines, the functionality of the application has been decomposed in seven services, as shown in Figure 1. The *orchestration* service receives the requests from the user interface, and passes them onto the appropriate service. When users upload simulation projects, or ask to edit them, the orchestration service just stores them in the *storage service*. However, when users request to run one single simulation once or several times, or when they request a parameter sweep, the orchestration service publishes the corresponding number of simulation tasks in the *queue service*. These simulation tasks will be accomplished by the *simulation service*, that takes input files and writes results

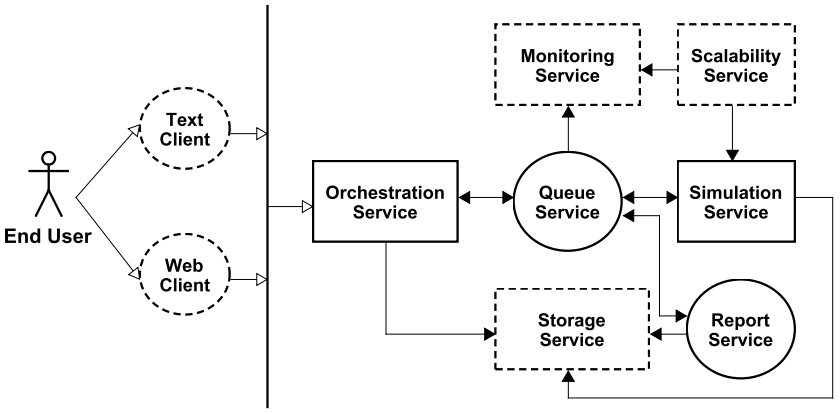


Figure 1: DNSE3 service oriented architecture. Other clients may be developed to interact with DNSE3 though the orchestration service. Dashed boxes represent services that can be reused from the cloud infrastructure. Solid circles are services used by DNSE3 but likely to be reused in other applications, while solid boxes correspond to services boxes are services very specific to DNSE3.

in the storage service. In order to cope with the existing workload, there may exist multiple running replicas of the simulation service at a given time. The actual number of instances is determined by administrator defined rules evaluated by the *scalability service* using the metrics that the queue service publishes in the *monitoring service*. Finally, the *report service* takes simulation results from the storage service and process them according to the instructions specified in the simulation project, in order to produce some reports for the user.

It should be noted that not all services need to be developed. Both public clouds and cloud middleware used to set up private clouds include storage, monitoring and scalability services that can be reused, calling their APIs. It is also noteworthy that though the orchestration service is quite problem specific, and the simulation service is a wrapper for ns-3, the queue and report services can be easily reused.

### *3.4. Job scheduling*

Job scheduling is not a simple task in a distributed system in which the number of *workers* (in this case, simulation services) varies dynamically. A centralized scheduler should be aware of all available workers and their status. To do so, the workers should publish regular updates to the scheduler, which increases its workload and sets up a bottleneck on the communication channel [34], while the scheduler should be able to cope with inconsistencies in the information.

Instead, a distributed scheduling approach based on work stealing [35] can be more suitable: the queue service maintains a list of available jobs (simulations) to be taken; when a new simulation service is launched, or when a running one finishes its previous work, it requests one or several works from the queue service. Therefore, every simulation service must know where to find the queue service, but the latter is not aware of how many instances of the former are running at a given time. When a worker takes a job it simply marks it as "in progress" in the queue, downloads its description, goes to the storage service to retrieve the input files, adequately configures and runs ns-3, and then uploads the outputs back to the storage service and reports the queue service that the work is "finished" before asking for a new one. To avoid that a crashed worker leaves a work unfinished, each of them must periodically refresh the status of the work in the queue service. Simulations not updated for a while are offered again to the next requesting worker.

It is noteworthy that, since workers will only ask for the *next job*, the queue service can implement ordering algorithms transparently to the workers. In our design, to achieve a fair allocation of resources among users, we have used a two-level round robin ordering of works: first by users and then by publication date. Every time a worker requests a simulation, the queue moves to the next user and then to the next available work. The benefits of this approach are twofold: no user can leave others without computational resources, and race conditions are avoided since every request for the *next job* will return a different job, even if the previous requester has not yet committed to take the job (unless,

of course, there is only one job in the queue).

Finally, it should be noted that this scheduling approach makes scalability much easier. When launching a new simulation service, the only caution to be taken is that it should know where to find the queue service to ask for a simulation work. When shutting one down, it would be wise to wait until it finishes its current work, though if not the queue will notice the work is idle and will announce it to the next worker. Significantly, the queue service is not at all affected by the increase or decrease of simulation services.

### *3.5. Workload metrics and scalability policies*

Automatic scalability is one main reason behind the proposal of DNSE3. The system should be able to dynamically accommodate the amount of computational resources to the existing workload, so that response times are kept low while making a sensible utilization of the resources. A decentralized scheduling based on work stealing facilitates this, since it suffices to start or stop simulation services following on *scalability rules* that evaluate *metrics* published by the monitoring service.

Since in DNSE3 scalability is to be achieved by increasing or decreasing the number of simulation services, it seems reasonable to use the number of works in the queue as an estimation of the running or yet unassigned workload. Other, more fine grained metrics, like the expected simulation duration or the number of simulation steps could be derived, though they are sometimes difficult to estimate and are more specific to the underlying ns-3 simulator. Besides, it should be noted that when several workers are running, a mix of short and long simulations can be dealt with efficiently, yielding short average response times. Nevertheless, one actual concern of basing scalability on the number of works in the queue appears when setting the rule thresholds, since they may depend on the average duration of the works (e.g. having 100 long pending simulations may call for scaling up, while if they were 100 short works existing resources could be enough).

Therefore, a scalability policy consisting of rules determining when to scale

up and down should be defined. The DNSE3 has been designed so that this can be done by the administrator of each DNSE3 deployment. Nevertheless, care should be taken in this process, considering that launching an instance (or shutting it down) takes a time that cannot be neglected. A new simulation service should be started only if it is expected that it will be useful for some time. Similarly, it should be stopped only when the remaining services have still a margin to accommodate moderate load increments. It is therefore inadvisable to use the same threshold to scale up and down, producing very sensitive rules that lead to frequent fluctuations in the number of services, yielding bad resource utilization, yet contributing little to improve response times. An example of a policy meeting this criteria is given later in section 4.2.

It is also wise to include a rule keeping a minimum number of instances running continuously, so that the system also achieves good response times for small workloads. In addition, to limit the bill charged for computational costs, an upper limit could also be set.

## 4. Prototype implementation

After having discussed the main design issues to build DNSE3 achieving self-scalability, the application need to be developed and deployed. This section briefly outlines some relevant decisions in this concern.

### *4.1. Technology decisions*

There are many popular public cloud providers, encompassing several levels of abstraction in what they virtualize, either Infrastructure, Platform, Software or even Container as a Service (IaaS, PaaS, SaaS and CaaS, respectively). When deploying an application that wraps desktop oriented third party software, such as ns-3, the IaaS approach allows to control the low level details of the execution environment. Besides, IaaS platforms are capable of monitoring both infrastructure and application level metrics to trigger scalability rules, and provide clear interfaces to do so. Finally, mature middleware to set up private

clouds offer IaaS, interestingly with a tendency to comply with Amazon Web Services (AWS), which makes it easier to deploy applications in an hybrid cloud.

For these reasons, DNSE3 has been developed for and deployed in an Openstack [21] infrastructure, calling its standard services though a REST API that is compatible with that of AWS. In particular, DNSE3 storage service is offered by Swift, the distributed object storage provided by Openstack, while the monitoring service is replaced by Ceilometer (that allows the publication of standard and user defined metrics) and the scalability service is implemented by Heat, which enables the user to define scalability policies called "stacks". The rest of the services have been developed in Java, making use of the Restlet framework [36] to expose and invoke REST interfaces. The requests and responses transport JavaScript Object Notation (JSON) documents, as it is a light yet powerful representation to exchange information, followed both by Openstack and AWS interface definitions.

### *4.2. Scalability policy*

The queue service publishes in Openstack's Ceilometer the occupation of the queue periodically, which is read (with possible different periodicity) by Heat in order evaluate rules determining whether to scale up (increment the number of simulation service instances) or down (decrement it). Other rules can also determine to truncate this number, so that resources are kept within certain limits. In this work, we propose the following rules:

- To scale up, `if` $\frac{tasks}{instances} > T$ `then` $instances := instances + 1$.
- To scale down, `if` $\frac{tasks}{instances} < \frac{T}{2}$ `then` $instances := instances/2$.
- The number of instances should always meet that $minVM < instances < maxVM$.

In the previous rules, $T$, $minVM$ and $maxVM$ are parameters that the administrator could fix considering the expected duration of the simulation and a compromise between performance and cost.

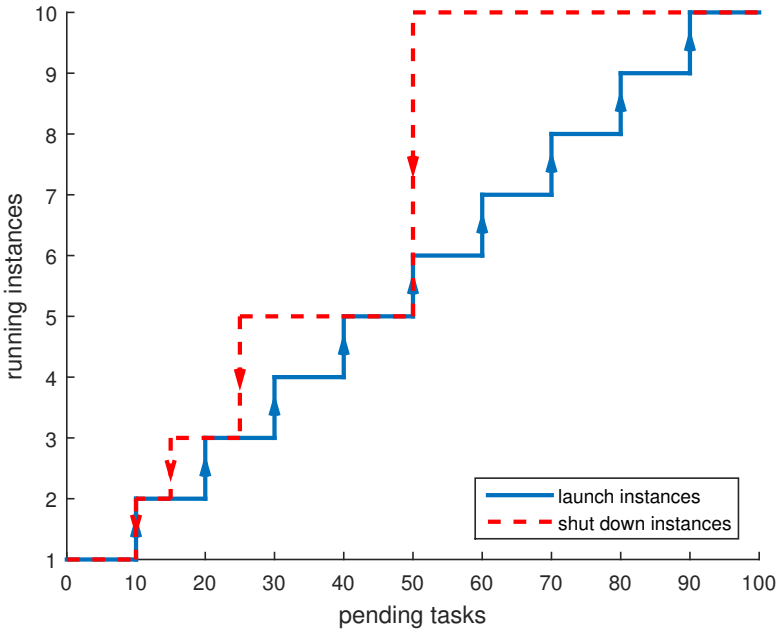


Figure 2: Actions triggered by scalability rules as the number of simulations in the queue vary, for $T = 10$, $minVM = 1$ and $maxVM = 10$. Solid lines correspond to scaling up, while dashed lines represent scaling down.

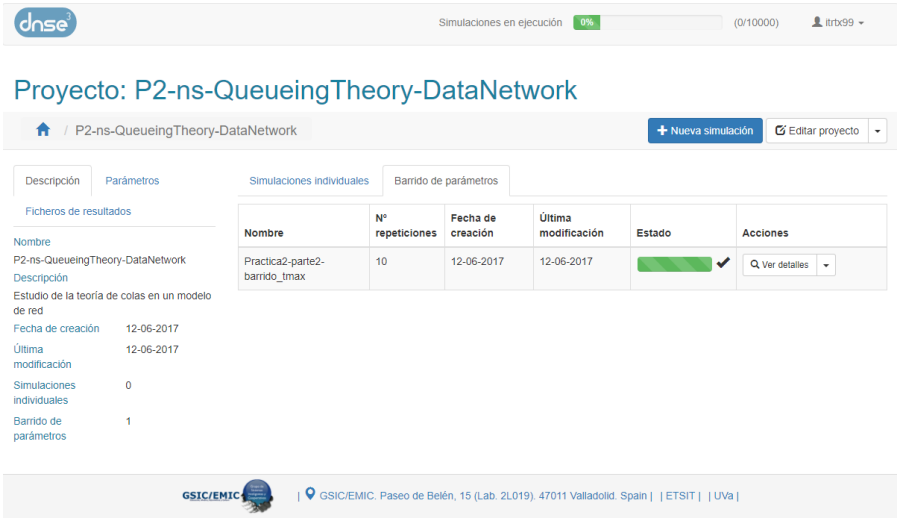


Figure 3: Snapshot of the web client graphical interface, when the user has demanded to run a parameter sweep and it has finished.

This scalability policy is illustrated graphically in Figure 2. It can be seen that the thresholds for scaling up and down are apart from each other. Along with the fact that rules are not evaluated continuously, but every time Heat reads the workload (in our prototype we choose 15 seconds), this policy is stable and will not produce strong fluctuation in the number of running instances.

### *4.3. User interface*

In the proposed architecture (see Figure 1) the orchestration service exposes a single entry point to the DNSE3. This service itself offers a REST API that can be called using any HTTP client (as a web browser, for example). Therefore, any third party can build a text or graphics based client that suits their needs and, in fact, during the development of the prototype, we have written our own command line clients. However, for the educational purposes that motivated the design of DNSE3, a visual web interface has been designed and implemented, as illustrated in Figure 3. After the user has logged in, he or she can see all the simulation projects, select one to see more details or edit it, start simulations, follow their progress, pause and resume then, analyze simple

results or retrieve data files to be used with external visualization tools, etc. This web application has been developed with the Bootstrap framework [37], in the search of compatibility with different browsers and devices, particularly smartphones and tables, which are popular among students.

### *4.4. Security management*

Though secondary to this research, there are also some security cautions in DNSE3. During its deployment, the orchestration service was configured to trust any request coming only from some particular clients, including the one installation of the web application, which was in turn responsible of credential checking. It validated credentials against an LDAP server existing in the laboratory used for the Computer Networks course assignments, so students did not need to be registered in the DNSE3 application, and then passed the user name along with any request to the orchestration service, who trusted requests came from that user. Clearly, this security policy can be fine when all parts of the application (including clients) belong to the same institution, but it is very weak otherwise. Despite of the importance of security when running cloud applications [38], the exploration of other mechanisms is beyond the scope of this paper.

## 5. System evaluation

A fully functional prototype of DNSE3 was developed as described above, and deployed in a private Openstack cloud. The evaluation carried out was twofold: on the one hand, experiments focused on performance were carried out using synthetic workloads, in order to assess scalability and compare DNSE3 with the software and hardware solutions used previously in the laboratory; on the other hand, usability was analyzed by asking real students to perform some tasks accessing DNSE3 user interface and evaluating their experience.

### 5.1. System performance

In order to evaluate scalability, several experiments were carried out using synthetic workloads (in the sense that all requests are performed automatically by a script, but it should be noticed the models simulated were those used in the real laboratory sessions, so that the average computation required for *one single* simulation is similar to that observed by the students). To evaluate the impact of increasing workload, the performance was evaluated for parameter sweeps resulting in 50, 100, 500, 1,000, 5,000 and 10,000 individual simulations. The first system to evaluate was the DNSE3, in which the scalability policy was as described in section 4.2, setting the scalability threshold $T = 10$ simulations per instance, the maximum number of simulation instances, $maxVM = 15$, while experiments were repeated for $minVM = 1$ and $minVM = 5$, to study its influence. To run the simulation service, small virtual machines were chosen, with a single virtual core and 2GB of RAM. Nevertheless, we explored if there were benefits in scheduling more than one job per worker ($N_{jobs}$) by repeating the experiments with $N_{jobs} = 1$ and $N_{jobs} = 2$. For comparison, we also evaluated a typical desktop computer available in the laboratory at the students' disposal, running a quad-core Intel(R) Core (TM) i5-2400 at 3,10 GHz with 4GB of RAM. For the latter, the task consisted just in a script launching the $N_{ps}$ simulations at once, so no overheads are involved. Additionally, all the experiments were carried out when no other users were connected and no heavy processes running, so competing workload can be neglected.

Following [11], all the experiments were repeated 5 times to detect outliers due to reasons external to the system under evaluation (e.g. unexpected competing workload from other sources) and to account for uncertainties (e.g. times to complete input/output operations). Table 1 shows the average response times for all the evaluated systems, for each of the experiments, while Figure 4 depicts the speedup of each configuration of DNSE3 with respect to the laboratory computer.

Observing the **most basic configuration** of DNSE3, starting with just one simulation service running ($minVM = 1$) that takes only one job at a

| System | Number of simulations | | | | | |
|---|---|---|---|---|---|---|
| | **50** | **100** | **500** | **1,000** | **5,000** | **10,000** |
| Laboratory machine | 88 | 177 | 889 | 1,809 | 9,444 | 17,330 |
| DNSE3, $minVM = 1$, $N_{jobs} = 1$ | 257 | 374 | 955 | 1,337 | 3,390 | 6,142 |
| DNSE3, $minVM = 1$, $N_{jobs} = 2$ | 224 | 314 | 726 | 1,076 | 3,204 | 5,967 |
| DNSE3, $minVM = 5$, $N_{jobs} = 1$ | 78 | 147 | 556 | 895 | 3,009 | 5,731 |
| DNSE3, $minVM = 5$, $N_{jobs} = 2$ | 62 | 121 | 464 | 712 | 2,774 | 5,522 |

Table 1: Response times (in seconds) for the four different configuration of DNSE3 and the laboratory machine used as reference.

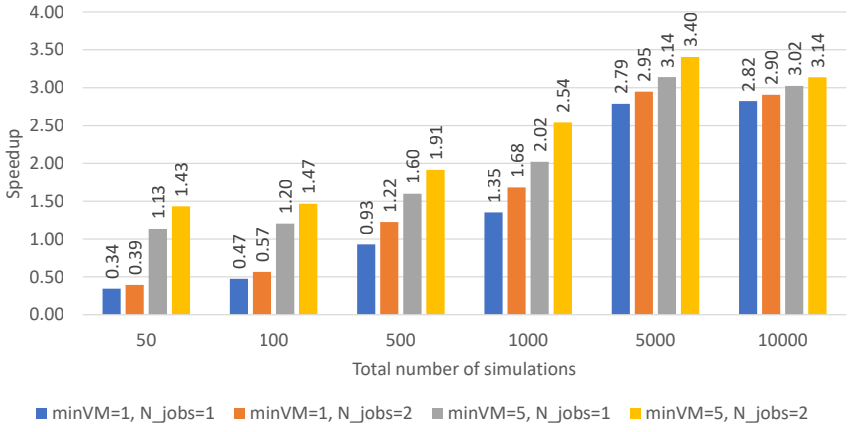


Figure 4: Speedup in performance (number of times faster) achieved by each of the configurations of DNSE with respect to the reference machine.

time ($N_{jobs} = 1$), it can be noticed that it does not improve performance for small parameter sweeps. In fact, for the smallest possible task which is running one single simulation (not shown in the table), the laboratory machine takes between 2 and 3 seconds (with the models used in these experiments), while DNSE3 takes up to 7. This is due to the fact that in DNSE3 the orchestration service publishes the work in the queue service and uploads input files into the storage service, then some simulation service picks up the work (polls are not continuous, so it may not be instantly), it downloads files from the storage service, compiles and calls ns-3, then uploads results to the simulation service and finally notifies the queue service of the completion of the work.

While this important overhead can be compensated by parallelism, scaling up does not make it on time to be be profitable if the parameter sweep is too short. As the number of requested simulations increase, the parallelism achieved by DNSE3 self-scaling allows to outperform its competitor to a point of being 2.8 times faster in the largest evaluated parameter sweep.

To deal with the limitations of DNSE3 in small tasks, parallelism could be exploited more aggressively, either by starting with a larger number of running simulation instances, or by allocating more than one job to each of the workers. As seen in figure 4, **starting with more simulation services already running** ($minVM = 5$), DNSE3 outperforms the competing machine even in the shortest tasks. As expected, it can parallelize tasks from the beginning, thus compensating the distribution overhead. It should be noted, however, that this comes at a price, since the five virtual machines must be continuously allocated. On the other hand, the effect of **running more than one job at each simulation service** ($N_{jobs} = 2$) has a more moderate, yet noticeable improvement. Remarkably, this is at no cost, since the committed resources are the same. One interesting issue is exploring the reasons for this, considering the virtual machines are single core. Clearly, if the simulation service was deployed in physical machines, when one process called to access input/output, the scheduler would keep the processor occupied with another one. In a virtualized environment, the hypervisor may decide to move into the host processor another virtual machine,

so the gains of this approach may depend on the actual load of the host machine rather than the virtual one.

Finally, we evaluated the addition of **both improvements** at the same time. As each one contributes to the speedup for different reasons, the benefit accumulates to some extent. Nevertheless, it can be seen that the speedup is not so noticeable for the 10,000-simulations parameter sweep (as compared to those with 5,000). This is because, in spite of the large workload in the system, the scalability rules keep the number of simulation instances under $maxVM = 15$. This fact suggest that, if performance is more important than costs, this limit should be removed.

### *5.2. System usability*

Besides shortening response times, to be used in the educational context the system should offer a good user experience. In order to evaluate usability, students already enrolled in a Computer Networks course in which they should use ns-3 to run parameter sweeps (creating scripts to generate the individual simulations, compile the model and call ns-3) were requested to participate in a experiment to evaluate DNSE3. Ten students volunteered to participate in the study. The experiment was self-guided with written instructions that introduced them quickly to DNSE3 main concepts, and then asked them to repeat two of the tasks they had already done during the regular laboratory sessions, this time using the new application. After that, their subjective perceptions of the previous tooling used in the laboratory (scripts and nd-3) and that of DNSE3 were surveyed.

To assess usability, the System Usability Scale (SUS) [39] was employed, in the version modified by [40], and translated into Spanish (the mother tongue of the students) by the researchers. This scale has been widely regarded as providing a quick yet consistent estimation of the usability, and has been broadly used to asses software in many domains including education [41, 42]. It consists of ten statements about the system under evaluation, alternatively positive and negative, to which the subject must express his or her degree of agreement with

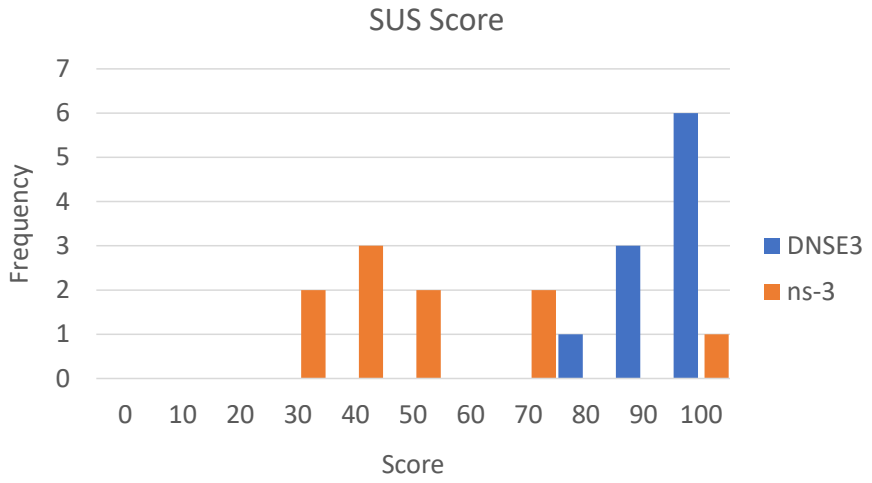


Figure 5: Distribution of the SUS scores in bins of 10, showing the perception of DNSE3 is consistently good, while the usage of scripts and ns-3 causes more diverse reactions.

a value from 1 to 5. Then an overall score is computed, between 0 and 100. [40] suggests not to report answers to each individual questions, since they are highly correlated, but just the SUS score, and proposes a rule of thumb to translate this score into a literal adjective describing the usability of the system.

In addition to the SUS survey, students were asked three more questions: to describe freely their most positive and their most negative perceptions of the systems, and their Likelihood To Recommend (LTR) the system to a user with such a need (i.e. another student taking the course), since several researchers have found a strong correlation with the SUS score [43].

The distribution of the scores is shown graphically in Figure 5. Overall, DNSE3 obtained a SUS score of 91.75 ("excellent" according to [40]) while the alternative software obtained only 47.75 ("poor"). The sample size does not allow to generalize these results (even if [44] reports some studies finding consisting results between small samples of 8-12 users and large samples). Nevertheless, it is worth mentioning that every student gave a higher score to DNSE3, and that scores to individual questions are quite uniform in the case of DNSE3 while

they exhibit a higher variance when referring to the scripts and ns-3 based solution. In addition, the LTR was 5 for the DNSE3 whilst 2.3 for the other tools, somehow confirming this promising result.

Finally, users made several suggestions in the free comments of the survey: some mentioned issues of the graphical interface that they did not like, others proposed new functionalities (e.g. to integrate a C++ editor in the interface to modify the simulation model without having to use a different tool). Interestingly, some pointed out as well the perception that the reduction of computation times helped them to complete the task more easily.

## 6. Conclusions and future work

Though parameter sweep simulations can significantly aid to understand the behavior of protocols in a computer network, they impose a computational cost that renders them unfeasible in many learning contexts. However, they feature high level parallelism that can be exploited to improve response times. In this sense, this paper has proposed DNSE3, a service oriented application to be deployed in an IaaS cloud computing platform in order to perform parameter sweep simulations using ns-3 as the underlying simulation engine. It features a decentralized scheduling approach with a work queue ordering the simulations so that resources are allocated fairly, and several independent simulation workers taking works from that queue. The number of simulation services is determined dynamically after analyzing scalability rules evaluated on application level workload metrics, thus achieving self-scalability.

The evaluation of DNSE3 has shown there is a significant improvement in performance for large parameter sweeps, even if the exploitation of parallelism is not too aggressive (starting with only one simulation instance that takes only one job at a time). A more advanced strategy yields much better results even for small tasks, though at the cost of having more resources permanently provisioned. It should be noted, however, that other parameters such as the interval between applications of the scalability policy, and that between polls

to take works made by simulation workers could also tuned to further improve performance. Moreover, both DNSE3 workload metrics and scalability rules can be defined by the administrator easily, opening the way to adjusting the behavior to simulation loads different from those evaluated in this paper.

Besides performance, usability was also studied with a set of students performing tasks with both DNSE3 and the more traditional approach used regularly in their laboratory assignments. Their preference for the new application was unanimous, with high scores for the System Usability Scale and the Likelihood To Recommend the system.

All in all, the new proposal shows promising benefits to introduce parameter sweeps in regular laboratory sessions for Computer Network courses. Moreover, it should be noted that although the user interface was designed with the learning context in mind, front-end clients can be easily replacea, making DNSE3 suitable for other purposes. Furthermore, while some of the services included in the DNSE3 architecture are quite problem specific, the overall design that enables automatic scalability is not, and it should not be difficult to refactor this architecture to achieve a self-scalable system for another domain with much process level parallelism.

## Acknowledgements

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness (grant TIN2014-53199-C3-2-R) and the Regional Government of Castilla y León (grant VA082U16, co-financed by the European Regional Development Fund, and grant VA277U14).

## References

[1] A. M. Law, W. D. Kelton, Simulation modeling and analysis, McGraw-Hill, New York, NY, USA, 1991.

[2] E. Weingartner, H. vom Lehn, K. Wehrle, A performance comparison of recent network simulators, in: IEEE International Conference on Communications, 2009, pp. 1–5. doi:10.1109/ICC.2009.5198657.

[3] The network simulator - ns-2 [online, visited on May 11, 2017].
URL http://www.isi.edu/nsnam/ns/

[4] The network simulator - ns-3 [online, visited on May 11, 2017].
URL https://www.nsnam.org/

[5] OMNeT++ discrete event simulator [online, visited on May 11, 2017].
URL https://omnetpp.org/

[6] Riverbed modeler [online, visited on June 28, 2017].
URL https://www.riverbed.com/es/products/steelcentral/steelcentral-riverbed-modeler.html

[7] C. Papadopoulos, J. Heidemann, Using ns in the classroom and lab, in: ACM SIGCOMM Workshop on Computer Networking, Pittsburgh, PA, USA, 2002, pp. 45–46.

[8] Z. A. Qun, W. Jun, Application of ns2 in education of computer networks, in: IEEE International Conference on Advanced Computer Theory and Engineering, 2008, pp. 368–372. doi:10.1109/ICACTE.2008.89.

[9] A. Wang, W. Jiang, Teaching wireless local area network course based on ns-3, in: International Symposium on Computer Network and Multimedia Technology, 2009, pp. 1–4. doi:10.1109/CNMT.2009.5374600.

[10] M. L. Bote-Lorenzo, J. I. Asensio-Pérez, E. Gómez-Sánchez, G. Vega-Gorgojo, C. Alario-Hoyos, A grid service-based distributed network simulation environment for computer networks education, Computer Applications in Engineering Education 20 (4) (2012) 654–665. doi:10.1002/cae.20435.

[11] R. Jain, The art of computer systems performance analysis: Techniques for experimental design, measurement, simulations and modelling, John Wiley & Sons, New York, NY, USA, 1991.

[12] S. Y. Wang, C. C. Lin, Y. S. Tzeng, W. G. Huang, T. W. Ho, Exploiting event-level parallelism for parallel network simulation on multicore systems, IEEE Transactions on Parallel and Distributed Systems 23 (4) (2012) 659–667. doi:10.1109/TPDS.2011.215.

[13] M. L. Bote-Lorenzo, Y. A. Dimitriadis, E. Gómez-Sánchez, Grid characteristics and uses: A grid definition, in: F. Fernández Rivera, M. Bubak, A. Gómez Tato, R. Doallo (Eds.), Grid Computing: First European Across Grids Conference, Santiago de Compostela, Spain, February 13-14, 2004. Revised Papers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 291–298. doi:10.1007/978-3-540-24689-3_36.

[14] OASIS Web Services Resource Framework (WSRF) [online, visited on June 28, 2017].
URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[15] L. M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, A break in the clouds: Towards a cloud definition, SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 50–55. doi:10.1145/1496091.1496100.

[16] L. M. Vaquero, L. Rodero-Merino, R. Buyya, Cloud scalability: building the millennium falcon, Concurrency and Computation: Practice and Experience 25 (12) (2013) 1623–1627. doi:10.1002/cpe.3008.

[17] Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges, Journal of Internet Services and Applications 1 (1) (2010) 7–18. doi:10.1007/s13174-010-0007-6.
URL http://dx.doi.org/10.1007/s13174-010-0007-6

[18] Google App Engine [online, visited on June 28, 2017].
URL https://cloud.google.com/appengine/?hl=es

[19] Microsoft azure [online, visited on June 28, 2017].
URL https://azure.microsoft.com/es-es/

[20] Amazon web services [online, visited on June 28, 2017].
URL https://aws.amazon.com/es/

[21] OpenStack Open Source Cloud Computing Software [online, visited on June 28, 2017].
URL https://www.openstack.org/

[22] Apache cloudstack open source cloud computing [online, visited on June 28, 2017].
URL https://cloudstack.apache.org/

[23] X. Zhou, H. Tian, Comparison on network simulation techniques, in: International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2016, pp. 313–316. doi:10.1109/PDCAT.2016.073.

[24] R. Fujimoto, Parallel and distributed simulation, in: Winter Simulation Conference, 2015, pp. 45–59.

[25] Q. Bragard, A. Ventresque, L. Murphy, Self-balancing decentralized distributed platform for urban traffic simulation, IEEE Transactions on Intelligent Transportation Systems 18 (5) (2017) 1190–1197. doi:10.1109/TITS.2016.2603171.

[26] Y. Cao, X. Jin, Z. Li, A distributed simulation system and its application, Simulation Modelling Practice and Theory 15 (1) (2007) 21 – 31. doi:10.1016/j.simpat.2006.09.010.

[27] D. Zehe, A. Knoll, W. Cai, H. Aydt, Semsim cloud service: Large-scale urban systems simulation in the cloud, Simulation Modelling Practice and Theory 58 (2015) 157 – 171. doi:10.1016/j.simpat.2015.05.005.

[28] C. Hüning, M. Adebahr, T. Thiel-Clemen, J. Dalski, U. Lenfers, L. Grundmann, Modeling & simulation as a service with the massive multi-agent system MARS, in: Agent-Directed Simulation Symposium, 2016, pp. 1–8.

[29] J. A. González-Martínez, M. L. Bote-Lorenzo, E. Gómez-Sánchez, R. Cano-Parra, Cloud computing and education: A state-of-the-art survey, Computers & Education 80 (2015) 132 – 151. doi:10.1016/j.compedu.2014.08.017.

[30] M. Behrisch, L. Bieker, J. Erdmann, D. Krajzewicz, SUMO – simulation of urban mobility, in: International Conference on Advances in System Simulation, 2011, pp. 63–68.

[31] S. Vinoski, REST eye for the SOA guy, IEEE Internet Computing 11 (1) (2007) 82–84. doi:10.1109/MIC.2007.22.

[32] V. Indhumathi, G. M. Nasira, Service oriented architecture for load balancing with fault tolerant in grid computing, in: IEEE International Conference on Advances in Computer Applications (ICACA), 2016, pp. 313–317. doi:10.1109/ICACA.2016.7887972.

[33] R. T. Fielding, Architectural syles and the design of network-based software architectures, Ph.D. thesis, University of California, Irvine, USA (2000).

[34] N. S. Arora, R. D. Blumofe, C. G. Plaxton, Thread scheduling for multiprogrammed multiprocessors, in: ACM Symposium on Parallel Algorithms and Architectures, 1998, pp. 119–129. doi:10.1145/277651.277678.

[35] R. D. Blumofe, C. E. Leiserson, Scheduling multithreaded computations by work stealing, J. ACM 46 (5) (1999) 720–748. doi:10.1145/324133.324234.

[36] Restlet framework [online, visited on June 28, 2017].
URL https://restlet.com/open-source/

[37] Bootstrap [online, visited on June 28, 2017].
URL http://getbootstrap.com/

[38] L. Coppolino, S. D'Antonio, G. Mazzeo, L. Romano, Cloud security: Emerging threats and current solutions, Computers & Electrical Engineer-

ing 59 (2017) 126 – 140. doi:https://doi.org/10.1016/j.compeleceng.2016.03.004.

[39] J. Brooke, SUS: A quick and dirty usability scale., in: P. W. Jordan, B. Thomas, I. L. McClelland, B. Weerdmeester (Eds.), Usability evaluation in industry, Taylor & Francis, London, UK, 1996, pp. 189–194.

[40] A. Bangor, P. T. Kortum, J. T. Miller, An empirical evaluation of the system usability scale, International Journal of Human & Computer Interaction 24 (6) (2008) 574–594. doi:10.1080/10447310802205776.

[41] H.-C. K. Lin, M.-C. Chen, C.-K. Chang, Assessing the effectiveness of learning solid geometry by using an augmented reality-assisted learning system, Interactive Learning Environments 23 (6) (2015) 799–810. doi:10.1080/10494820.2013.817435.

[42] A. Martin-Gonzalez, A. Chi-Poot, V. Uc-Cetina, Usability evaluation of an augmented reality system for teaching euclidean vectors, Innovations in Education and Teaching International 53 (6) (2016) 627–636. doi:10.1080/14703297.2015.1108856.

[43] J. R. Lewis, Usability: Lessons learned... and yet to be learned, International Journal of Human & Computer Interaction 30 (9) (2014) 663–684. doi:10.1080/10447318.2014.930311.

[44] J. Brooke, SUS: A retrospective, J. Usability Studies 8 (2) (2013) 29–40.

# Apéndice C

# Esquema XSD de definición de los metadatos de los proyectos de simulación

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3          targetNamespace="http://gsic.tel.uva.es/dnse3/spd_v0p9"
4          xmlns="http://gsic.tel.uva.es/dnse3/spd_v0p9"
5          elementFormDefault="qualified">
6
7          <xsd:element name="simulationpackage">
8                  <xsd:complexType>
9                          <xsd:sequence>
10                                 <xsd:element ref="description"
                                       minOccurs="1"
11                                         maxOccurs="1" />
12                                 <xsd:element ref="parametertypes"
                                       minOccurs="1"
13                                         maxOccurs="1" />
14                                 <xsd:element ref="outputfilestructures"
                                        minOccurs="1"
15                                         maxOccurs="1" />
16                                 <xsd:element ref="outputfiles"
                                       minOccurs="1"
17                                         maxOccurs="1" />
18                         </xsd:sequence>
19                 </xsd:complexType>
20         </xsd:element>
21
22         <xsd:element name="description">
23                 <xsd:complexType>
24                         <xsd:sequence>
25                                 <xsd:element name="title" type="xsd:
                                       string"
26                                         minOccurs="0" maxOccurs="1" />
27                                 <xsd:element name="textualdescription"
                                       type="xsd:string"
28                                         minOccurs="0" maxOccurs="1" />
```

```
29                      </xsd:sequence>
30              </xsd:complexType>
31      </xsd:element>
32
33      <xsd:element name="inputfiles">
34              <xsd:complexType>
35                      <xsd:sequence>
36                              <xsd:element name="basefile" type="xsd:
                                    string"
37                                      minOccurs="1" maxOccurs="1" />
38                              <xsd:element name="additionalfile" type
                                    ="xsd:string"
39                                      minOccurs="0" maxOccurs="
                                        unbounded" />
40                      </xsd:sequence>
41              </xsd:complexType>
42      </xsd:element>
43
44      <xsd:element name="parametertypes">
45              <xsd:complexType>
46                      <xsd:choice minOccurs="1" maxOccurs="unbounded"
                            >
47                              <xsd:element ref="stringtype" />
48                              <xsd:element ref="integertype" />
49                              <xsd:element ref="rationaltype" />
50                              <xsd:element ref="seedtype" />
51                      </xsd:choice>
52              </xsd:complexType>
53      </xsd:element>
54
55      <xsd:element name="stringtype">
56              <xsd:complexType>
57                      <xsd:sequence>
58                              <xsd:element name="possiblevalue" type=
                                    "xsd:string"
59                                      minOccurs="0" maxOccurs="
                                        unbounded" />
60                              <xsd:element name="defaultvalue" type="
                                    xsd:string" minOccurs="1"
61                                      maxOccurs="1" />
62                      </xsd:sequence>
63                      <xsd:attribute name="id" type="xsd:ID" use="
                            required" />
64              </xsd:complexType>
65      </xsd:element>
66
67      <xsd:element name="integertype">
68              <xsd:complexType>
69                      <xsd:sequence>
70                              <xsd:choice minOccurs="0" maxOccurs="1"
                                    >
71                                      <xsd:element name="greaterthan"
                                            type="xsd:integer" />
72                                      <xsd:element name="
                                        greaterthanorequalto"
```

```
73                                              type="xsd:integer" />
74                                  </xsd:choice>
75                                  <xsd:choice minOccurs="0" maxOccurs="1"
                                        >
76                                      <xsd:element name="lessthan"
                                            type="xsd:integer" />
77                                      <xsd:element name="
                                            lessthanorequalto"
78                                          type="xsd:integer" />
79                                  </xsd:choice>
80                                  <xsd:element name="defaultvalue" type="
                                        xsd:integer" minOccurs="1"
81                                      maxOccurs="1" />
82                          </xsd:sequence>
83                          <xsd:attribute name="id" type="xsd:ID" use="
                                required" />
84                  </xsd:complexType>
85          </xsd:element>
86
87          <xsd:element name="rationaltype">
88                  <xsd:complexType>
89                          <xsd:sequence>
90                                  <xsd:choice minOccurs="0" maxOccurs="1"
                                        >
91                                      <xsd:element name="greaterthan"
                                            type="xsd:float" />
92                                      <xsd:element name="
                                            greaterthanorequalto"
93                                          type="xsd:float" />
94                                  </xsd:choice>
95                                  <xsd:choice minOccurs="0" maxOccurs="1"
                                        >
96                                      <xsd:element name="lessthan"
                                            type="xsd:float" />
97                                      <xsd:element name="
                                            lessthanorequalto"
98                                          type="xsd:float" />
99                                  </xsd:choice>
100                                 <xsd:element name="defaultvalue" type="
                                        xsd:float" minOccurs="1"
101                                     maxOccurs="1" />
102                         </xsd:sequence>
103                         <xsd:attribute name="id" type="xsd:ID" use="
                                required" />
104                 </xsd:complexType>
105         </xsd:element>
106
107         <xsd:element name="seedtype">
108                 <xsd:complexType>
109                         <xsd:attribute name="id" type="xsd:ID" use="
                                required" />
110                 </xsd:complexType>
111         </xsd:element>
112
113         <xsd:element name="outputfilestructures">
```

```
114                 <xsd:complexType>
115                         <xsd:sequence>
116                                 <xsd:element ref="filestructure"
                                        minOccurs="0"
117                                             maxOccurs="unbounded" />
118                         </xsd:sequence>
119                 </xsd:complexType>
120         </xsd:element>
121
122         <xsd:element name="filestructure">
123                 <xsd:complexType>
124                         <xsd:sequence>
125                                 <xsd:element name="outputvar" type="xsd
                                        :string"
126                                             minOccurs="1" maxOccurs="
                                                unbounded" />
127                         </xsd:sequence>
128                         <xsd:attribute name="id" type="xsd:ID" use="
                                required" />
129                         <xsd:attribute name="ismultilinefile" type="xsd
                                :boolean"
130                                 use="required" />
131                 </xsd:complexType>
132         </xsd:element>
133
134         <xsd:element name="outputfiles">
135                 <xsd:complexType>
136                         <xsd:sequence>
137                                 <xsd:element ref="tabbedfile" minOccurs
                                        ="0"
138                                             maxOccurs="unbounded" />
139                                 <xsd:element name="tracefile" type="xsd
                                        :string"
140                                             minOccurs="0" maxOccurs="
                                                unbounded" />
141                                 <xsd:element ref="resultfile" minOccurs
                                        ="0"
142                                             maxOccurs="unbounded" />
143                         </xsd:sequence>
144                 </xsd:complexType>
145         </xsd:element>
146
147         <xsd:element name="tabbedfile">
148                 <xsd:complexType>
149                         <xsd:simpleContent>
150                                 <xsd:extension base="xsd:string">
151                                         <xsd:attribute name="
                                                filestructureref"
152                                                 type="xsd:IDREF" use="
                                                    required" />
153                                 </xsd:extension>
154                         </xsd:simpleContent>
155                 </xsd:complexType>
156         </xsd:element>
157
```

```
158         <xsd:element name="resultfile">
159                 <xsd:complexType>
160                         <xsd:simpleContent>
161                                 <xsd:extension base="xsd:string">
162                                         <xsd:attribute name="
                                                filestructureref"
163                                                 type="xsd:IDREF" use="
                                                    required" />
164                                 </xsd:extension>
165                         </xsd:simpleContent>
166                 </xsd:complexType>
167         </xsd:element>
168
169 </xsd:schema>
```

# Apéndice D

# Informes generados por el servicio de informes

En este apéndice se incluyen los ficheros generados tras la ejecución de una simulación de barrido de parámetros, donde se ha fijado el valor del parámetro `lambda=9` y el parámetro `meanPacketSize` toma valores entre 100 y 1000 a incrementos de 100. La única variable recuperada ($Q$) está presente en la salida estándar y se almacena dentro de `output_Q.csv`. Cada uno de estos experimentos se ha repetido 10 veces.

## D.1. lambda.csv

```
9
9
9
9
9
9
9
9
9
9
```

## D.2. meanPacketSize.csv

```
100.0
200.0
300.0
400.0
500.0
600.0
700.0
800.0
900.0
1000.0
```

## D.3. output_Q.csv

```
1  0.0079926, 0.0074142, 0.00760814, 0.00734383, 0.00737626, 0.00757281,
       0.00733701, 0.00784117, 0.00754265, 0.00751963
2  3.25417, 3.31085, 3.28757, 3.02733, 3.08531, 2.90903, 3.46465, 3.03572,
        3.27373, 3.19016
3  0.0291022, 0.0285837, 0.0299734, 0.0294431, 0.0288231, 0.0297637,
       0.0281689, 0.0325972, 0.0323474, 0.0295066
4  0.0779779, 0.0760192, 0.0775408, 0.0742644, 0.0719787, 0.0745973,
       0.0763761, 0.077013, 0.0772453, 0.0825875
5  0.164441, 0.169009, 0.169445, 0.152593, 0.168135, 0.168406, 0.163548,
       0.14245, 0.149811, 0.160593
6  0.299508, 0.286894, 0.306833, 0.312658, 0.286368, 0.299452, 0.28638,
       0.273318, 0.29553, 0.320821
7  0.505875, 0.501331, 0.503945, 0.525217, 0.511526, 0.4985, 0.525596,
       0.482389, 0.506853, 0.493999
8  0.825197, 0.785364, 0.804, 0.83127, 0.75028, 0.829569, 0.813859,
       0.81888, 0.786139, 0.807292
9  1.29714, 1.27501, 1.31268, 1.31361, 1.24193, 1.32756, 1.31099, 1.25704,
        1.36927, 1.37231
10 2.09921, 1.84667, 2.04457, 2.11798, 1.95189, 1.98966, 1.96416, 1.89251,
        2.01857, 2.0238
```

# Apéndice E

# Plantillas de definición de la política de escalado del DNSE3

## E.1. dnse3.yaml

```
1  heat_template_version: 2015-04-30
2
3  description: Heat template for scaling the simulation service
4
5  parameters:
6      flavor:
7          type: string
8          label: Flavor used by the instances
9          description: Flavor used by the instances
10         default: m1.small
11     key:
12         type: string
13         label: SSH key
14         description: SSH key
15         default: dnse3
16     private_network:
17         type: string
18         label: Network to connect the instances
19         description: Network to connect the instances
20         default: dnse3-pri-net
21     queue_address:
22         type: string
23         label: Queue Server IP address
24         description: Queue Server IP address
25         default: 192.168.50.87:8081
26     T:
27         type: number
28         label: Scale up threshold
29         description: Scale up threshold
30         default: 10
31     T_half:
32         type: number
33         label: Scale down threshold
34         description: Scale down threshold
```

```
35          default: 5
36      userID:
37          type: string
38          label: User ID
39          description: User ID
40      password:
41          type: string
42          label: User password
43          description: User password
44          hidden: true
45
46
47  resources:
48      instance_group:
49          type: OS::Heat::AutoScalingGroup
50          properties:
51              min_size: 1
52              max_size: 15
53              desired_capacity: 1
54              cooldown: 60
55              resource:
56                  type: serverDefinition.yaml
57                  properties:
58                      queue_address: {get_param: queue_address}
59                      userID: {get_param: userID}
60                      password: {get_param: password}
61                      flavor: {get_param: flavor}
62                      key: {get_param: key}
63                      private_network: {get_param: private_network}
64
65      scale_up:
66          type: OS::Heat::ScalingPolicy
67          properties:
68              adjustment_type: change_in_capacity
69              auto_scaling_group_id: {get_resource: instance_group}
70              scaling_adjustment: 1
71
72      scale_down:
73          type: OS::Heat::ScalingPolicy
74          properties:
75              adjustment_type: percent_change_in_capacity
76              auto_scaling_group_id: {get_resource: instance_group}
77              scaling_adjustment: '-50'
78
79      queue-alarm-high:
80          type: OS::Ceilometer::Alarm
81          properties:
82              meter_name: dnse3.queue.simulation.tasks_per_instance
83              threshold: { get_param: T}
84              alarm_actions:
85                  - { get_attr: [scale_up, alarm_url]}
86              comparison_operator: gt
87              description: Alarm to generate new instances
88              evaluation_periods: 1
89              statistic: avg
```

```
90              period: 15
91
92      queue-alarm-low:
93          type: OS::Ceilometer::Alarm
94          properties:
95              meter_name: dnse3.queue.simulation.tasks_per_instance
96              threshold: { get_param: T_half}
97              alarm_actions:
98                  - { get_attr: [scale_down, alarm_url]}
99              comparison_operator: lt
100             description: Alarm to remove instances
101             evaluation_periods: 1
102             statistic: avg
103             period: 15
104
105 outputs:
106     scale_up_url:
107         description: Webhook of the Scaling Up Policy
108         value: {get_attr: [scale_up, alarm_url]}
109     scale_down_url:
110         description: Webhook of the Scaling Down Policy
111         value: {get_attr: [scale_down, alarm_url]}
112     current_size:
113         description: Number of active instances
114         value: {get_attr: [instance_group, current_size]}
```

## E.2. serverDefinition.yaml

```
1  heat_template_version: 2015-04-30
2
3  description: Definition of the server configuration
4
5  parameters:
6    queue_address:
7      type: string
8      label: Queue Server IP address
9      description: Queue Server IP address
10   userID:
11     type: string
12     label: User ID
13     description: User ID
14   password:
15     type: string
16     label: User password
17     description: User password
18     hidden: true
19   flavor:
20     type: string
21     label: Flavor used by the instances
22     description: Flavor used by the instances
23   key:
24     type: string
25     label: SSH key
26     description: SSH key
```

```
27    private_network:
28      type: string
29      label: Network to connect the instances
30      description: Network to connect the instances
31      default: dnse3-pri-net
32
33  resources:
34    server:
35      type: OS::Nova::Server
36      properties:
37          image: dnse3-20170323
38          key_name: {get_param: key}
39          flavor: {get_param: flavor}
40          networks:
41            - network: {get_param: private_network}
42          user_data:
43            str_replace:
44              template: |
45                #!/bin/bash
46                echo "Running Simulation Client"
47                /home/ubuntu/Simulation/simulation.sh $queueAddress
                      $userID $passAccess
48                echo "Simulation client booted"
49              params:
50                $queueAddress: {get_param: queue_address}
51                $userID: {get_param: userID}
52                $passAccess: {get_param: password}
53          user_data_format: RAW
```