Noname manuscript No.
(will be inserted by the editor)

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

Sergio Serrano-Iglesias · Eduardo Gómez-Sánchez · Miguel L. Bote-Lorenzo · Juan I. Asensio-Pérez · Manuel Rodríguez-Cayetano

Received: date / Accepted: date

Abstract While parameter sweep simulations can help undergraduate students and researchers to understand computer networks, their usage in the academia is hindered by the significant computational load they convey. This paper proposes DNSE3, a service oriented computer network simulator that, deployed in a cloud computing infrastructure, leverages its elasticity and pay-per-use features to compute parameter sweeps. The performance and cost of using this application is evaluated in several experiments applying different scalability policies, with results that meet the demands of users in educational institutions. Additionally, the usability of the application has been measured following industry standards with real students, yielding a very satisfactory user experience.

Keywords computer networks simulation · automatic scalability · cloud computing applications

This work has been partially funded by the Spanish State Research Agency and the European Regional Development Fund (grants TIN2014-53199-C3-2-R and TIN2017-85179-C3-2-R) and the Regional Government of Castilla y León (grant VA082U16, co-financed by the European Regional Development Fund).

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

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

1 Introduction

Computer network simulation allows to evaluate protocols or network configurations that cannot be deployed in reality with ease or within a reasonable budget [29, 51], and to explore extreme traffic situations which are rare but of critical impact [51]. Besides researchers or standard proponents, simulation is often used in education because 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, often by proposing students to deal with one or just a few given network configurations (see for example the experiences reported in [38], [40] or [49]). 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 instance, [9] 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, this experiment demands a parameter sweep simulation that can be explored as a $2^k r$ factorial design [28, chapter 18]. With typical desktop computers available in educational laboratories, the response times would be very high, rendering the learning scenario unfeasible in practice.

In spite of the high computational load of a parameter sweep study, it can be decomposed into a (normally quite large) number of *independent* simulations, each of them evaluating the same model with a different set of parameters. Hardware solutions that exploit thread or process level parallelism can bring down simulation times [50], though to achieve sensible reductions a significant amount of hardware is needed, which normally

is beyond the budget of educational institutions, especially if this hardware only serves a few experiments in very specific courses and will thus remain unused most of the time. Instead, a shared infrastructure that may scale up and down easily would fit much better to these intermittent needs of computation. Indeed, DNSE [9] exploits a computational grid infrastructure based on Globus Toolkit 4 [18] to carry out parameter sweeps. With a service oriented design, simulation services in charge of running ns-2 [34] models can be replicated as needed in existing resources in the virtual organization (a "federations of resources" spanning multiple administrative domains). However, adding or removing computation nodes to the virtual organization requires manual intervention of the administrator. With a dynamic load, this implies either under or overprovisioning. Moreover, low adoption of service oriented grids (in particular one based on the Web Services Resource Framework, WSRF [36], with no updates or events since the publication of the standard in 2006) and Globus Toolkit (with support discontinued in January 2018) hinder the utility of DNSE.

On the contrary, the cloud computing paradigm has matured quickly with the consolidation of prominent public cloud platforms, such as Google App Engine [23], Microsoft Azurre [33] or Amazon Web Services (AWS) [1], and also robust middleware to set up private clouds, like OpenStack [37] or Cloudstack [3]. This paradigm allows on demand virtualized resource provisioning under a pay-per-use business model without upfront investment [47], where resources can be reserved and released without human intervention through user defined scalability rules [46], yielding virtually infinite scalability of an application at reasonable costs [53]. Though this fact makes cloud computing very appealing for scientific, performance-demanding applications [41], such as simulators, the usage of cloud computing in the educational realm is very much limited to ready Software-as-a-Service applications [22, 44].

This paper leverages the flexible scalability and cost model of the cloud to propose DNSE3 (*Distributed Network Simulation Environment 3*), an application conceived for education that allows students to define, run and analyze both single simulations and parameter sweeps. This new system evolves DNSE [9], making use of ns-3 [35], which allows embedding real network stack code in simulations [35] and brings much better performance in the simulation of wireless networks [54]. Moreover, a service oriented architecture is proposed for the application aimed at exploiting cloud features to achieve self-scalability for improved performance at reasonable costs. In particular, it uses the cloud provisioning mechanisms to automatically increment or reduce the computational resources used to run simulation services. These services take simulation tasks from one single pool using a work stealing approach, to avoid the need to know the existing simulation services in such dynamic environment. To govern how aggressively DNSE3 recruits or frees resources in order to adapt to existing workloads, a scalability policy must be defined consisting on rules that evaluate application level metrics and determine when and how much to scale up or down. This paper also evaluates several scalability policies that scale towards keeping the amount of pending tasks per worker at a certain level, presenting DNSE3 results in performance and cost when using these policies, and reflecting on their adequacy to achieve a desired response time. In addition, the usability of the developed application is tested with users from the domain (students).

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 functional and non functional requirements in the design of DNSE3. Section 4 presents the service oriented architecture, with detailed discussion on the decisions that drive automatic scalability, and describes how this architecture is deployed in a private Openstack architecture, discussing which services can be reused from the infrastructure and which cannot. Section 5 evaluates DNSE3 performance and cost, discussing the effect of several scalability policies, while section 6 very briefly reports the findings on application usability. 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 [19]. 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 [19]. A somewhat different approach of parallelization delivers separate but dependent simulation models to each of the computing elements (the *logical processes*, LP). For instance, [50] 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, [10] 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 beforehand, 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 [9] 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. [15] 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 [20]. SEMSim [52] 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. [26] claims to follow a similar approach, though the models they simulate are not restricted to a particular domain. A similar architecture is proposed by [25] for a different application: searching for asteroids in chunks of astronomical data processed in parallel by different virtual machines in an Openstack private cloud. Though the performance is evaluated for different scales, the number of virtual machines is fixed before each of the experiments.

Finally, cloud computing has also been used to support education in many ways [22,44], among them supporting simulation. The *Cloud-based collaborative and Scale-up Modelling and Simuation Framework for STEM Education* (C^2SuMo) [13] 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 [6] is launched for each user, though the work demanded by an individual user is not actually parallelized.

3 Application requirements

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.

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 through the DNSE3 graphical interface. Note that, unlike in [38], 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.

4 Application architecture and deployment

Once the desired functional and non-functional requirements have been identified, this section first describes a Service Oriented Architecture (SOA) [48] application architecture that fulfills them. A particular emphasis is given to the decisions on task scheduling that facilitate scalability. While the architecture is devised first independently of the deployment technology, the section discusses later which services of Openstack and AWS could be reused to deploy DNSE3 in either a private or a public cloud.

4.1 Architectural design

The DNSE3 architecture has been designed following the principles of Service Oriented Architectures (SOA) [48], 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 [27]. 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 [17], 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 set of 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 [17]. 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 [48].

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 them, the orchestration service takes this job, which publishes each of the individual simulation tasks in the *queue service*. These tasks be accomplished one by one by the *simulation service*, that takes input files and writes results 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 automatically by parametric rules evaluated by the *scalability service* using the metrics that the queue service publishes in the *monitoring service*. The scalability service is thus in charge of launching or releasing virtual machines that run simulation services. 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.

4.2 Task scheduling

Task scheduling is not simple in a distributed system in which the number of *workers* (in this case, simulation services) varies dynamically. A centralized sched-

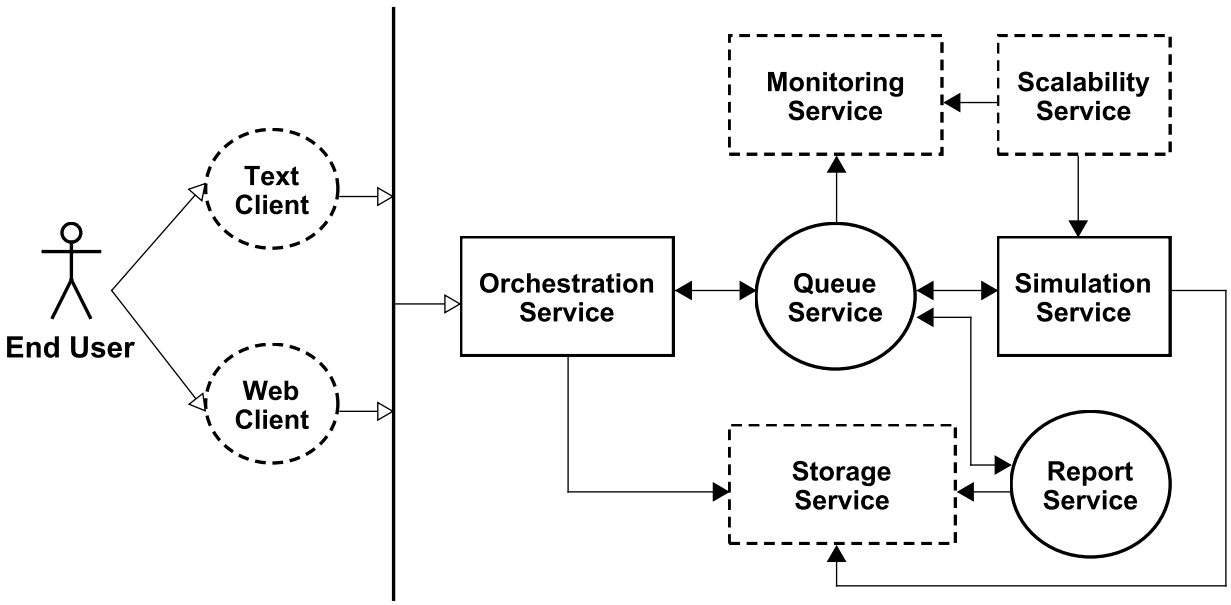

Fig. 1 DNSE3 service oriented architecture. Other clients may be developed to interact with DNSE3 though the orchestration service. Dashed boxes represent services that, in our deployment, we decided to reuse from Openstack infrastructure. Solid circles are services used by DNSE3 but likely to be reused in other applications, while solid boxes correspond to services boxes are services very specific to DNSE3.

uler 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 [4], while the scheduler should be able to cope with inconsistencies in the information. On the contrary, a fully distributed work stealing approach (in which each worker has its own task pool) can increase communication [7] and requires that workers advertise in a common bus when they come up and down.

Instead, work stealing with a single task pool can be more suitable: the queue service maintains a list of available tasks (simulations) to be taken; when a new simulation service is launched, or when a running one finishes its previous task, it requests one or several other tasks 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. In DNSE3, when a worker takes a task 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 task is "finished" before asking for a new one. To avoid that a crashed worker leaves a task unfinished, each of them must periodically refresh their status in the queue service. Simulation tasks that have not been updated for a while are offered again to the next requesting worker.

It is noteworthy that, since workers will only ask for the *next task*, 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 tasks: 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 task. The benefits of this approach are twofold: no user can leave others without computational resources (even if the scalability service will recruit more after a while), and race conditions are avoided since every request for the *next task* will return a different one, even if the previous requester has not yet committed to take the task (unless, of course, there is only one task 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 task. When shutting one down, it would be wise to wait until it finishes its current task, though if not the queue will eventually notice the task is idle and will announce it to the next worker. Signif-

icantly, the queue service is not at all affected by the increase or decrease of simulation services.

4.3 Deployment

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, we have decided to deploy DNSE3 in an Openstack infrastructure. We discuss next how we have decided to map DNSE3 services into Openstack standard services and, for the shake of completeness, we also comment what choices would be made if eventually deploying onto AWS public cloud. Interestingly, Openstack's REST API is designed to be compatible with that of AWS, and therefore the calls that application services make to the infrastructure should (ideally) not be changed if migrating between the two.

Concerning DNSE3 storage service, it can be offered by Swift or S3, the distributed object storage services provided by Openstack and AWS, respectively. Both allow to handle objects efficiently, and support a naming system to structure them hierarchically. Simulation projects, individual simulations, simulation models, output results and traces, and reports are all treated as objects with the associated information.

The monitoring service requires that the queue service is able to publish metrics, and that they reach the scalability service. In Openstack, this is achieved by Ceilometer (the telemetry data collection service, which allows the publication of standard and user defined metrics) and Aodh (the telemetry alarming service, which reads these metrics and triggers alarms). In AWS, the CloudWatch service would satisfy both needs.

With regards to DNSE3 scalability service, its functionality is offered in Openstack by Heat, which allows the user to publish a *stack*, i.e. a file that describes the infrastructure (machine sizes, images, etc) for an application, with associated resources that include potential scaling groups. Therefore, though Heat has a much broader scope, it can implement DNSE3 scalability service. It should also be noted that the only benefit of the self-scalability mechanism in being offered as a service is that policies can be managed from any REST client, but in DNSE3 architecture no application service will call the scalability service API. As for AWS, similar arguments would support the choice of the Auto Scaling service to endow DNSE3 with self-scaling mechanisms.

Finally, we have considered Openstack's Zaqar and AWS Simple Queue Service (SQS) to implement DNSE3 task queue service, though we found more convenient to develop it from scratch. Indeed, neither Zaqar nor SQS guarantee an allocation of tasks to workers considering a fair distribution among the users. SQS allows to differentiate messages by *group id* in FIFO queues, but while messages with the same group id are processed in strict order, there is no control on how SQS dispatches messages with different group id. Of course, it could be argued that, with virtually infinite scalability, sharing resources among users does not seem critical. Nevertheless, the actual extent of resources might be limited by hardware (in a private cloud) or budget (with a public provider). But even if this is not the case, at a point in time resources can be scarce for the existing workload until the self-scalability mechanism provision some more. Therefore, we have decided to keep this requirement to avoid that one student launches a large parameter sweep job and noticeably affect the performance perceived by other students. Another reason not to reuse Zaqar or SQS for DNSE3 queue service is that the task (or message) lifecycle is understood differently. In the former two, the message should be deleted from the queue by the worker that has processed it. In DNSE3, we considered more convenient that the worker marks the task as finished in the queue service, which in turn notifies the orchestration service that can thus monitor the progress of a job (and inform the end user). The orchestration service will delete from the queue all tasks of a given job when it is complete.

The rest of the services have been developed in Java, making use of the Restlet framework [42] 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.

5 Performance evaluation

DNSE3 is proposed to leverage the self-scalability and pay-per-use features of the cloud in order to compute parameter sweeps in reduced response times at reasonable costs, that can meet the needs of an educational

use. With this aim, this section discusses the choice of workload metrics and the design of scalability rules, proposing several policies, and then evaluates DNSE3 performance and resource utilization when dealing with parameter sweep simulations of different nature, reflecting on the effect of the chosen scalability policy.

5.1 Scalability design

DNSE3 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. The proposed task scheduling mechanism facilitates this, since it suffices to start or stop simulation services following *scalability rules* that evaluate *metrics* published by the monitoring service. Concerning the latter, current utilization of resources (e.g. CPU percentage) is often used to auto-scale web applications with rather unpredictable traffic patterns [45], while the amount of unassigned workload serves better for batch scientific computing, as is this case. As the objective of scalability should be to dimension the number of workers keeping each of them with a reasonable occupation, while achieving low aggregated response times, a good candidate metric is the number of pending tasks in the queue relative to the amount of active workers [21,2]. Other, more fine grained metrics, like the expected simulation duration or the number of simulation steps could have also be explored. Nevertheless, even if they relate more easily to the response time, they are far more difficult to estimate and are more specific to the underlying ns-3 simulator and to the specific features of the model being simulated.

Once an appropriate metric is chosen, scalability rules must be defined. It should be noted that, at the moment, Openstack supports only what AWS calls *simple rules* [2], i.e. rules that cannot be evaluated again until the effect of the last evaluation (starting or stoping new instances) is complete, while neither *step* nor *tracking* rules [2] are supported. Normally, rules evaluate that the chosen metric is within a predefined range, and change the amount of running instances by an absolute or relative increment (or reduction) or by setting a given number. Care should be taken in this process, considering that launching an instance (or shutting it down) takes some 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. Finally, 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. Similarly, to limit the bill charged for computational costs, an upper limit can also be set.

Following these guidelines, we have decided to explore several scalability policies, in order to reflect on their impact in performance and cost, as well as on how to choose thresholds. The proposed policies are described in detail in Table 1. The first two policies are evaluated as a reference of limits in cost and performance: P_{none} keeps the number of running instances to $minVM$, while P_{all} scales up to $maxVM$ as soon as there are pending tasks in the queue, and comes back to $minVM$ when the queue empties. We then propose to evaluate a conservative stepped policy, P_{cons}, which is often reported in the literature [45,43,14], in which the number of instances is incremented at most by one in each evaluation. Note that the thresholds to scale up and down are different, in order to avoid continuous fluctuations in the number of instances. Also observe that this policy will converge to a stable deployment when the ratio of tasks per instance is between $T/2$ and T, so in the end it will produce a result similar to a tracking policy, though expectedly much more slowly. We therefore propose to evaluate a more aggressive set of rules, P_{aggr}, in which the number of new resources is in proportion both to already running instances and distance to the target (some examples of this approach can also be found in the literature [24,16]). Finally, it should be noted that rules #11 and #12 are present in all the policies to keep the number of instances bounded (in general, lower bounds are chosen to satisfy a minimum desired throughput, while upper bounds are given by cost).

5.2 Experimental setup

As mentioned above, a fully functional prototype of DNSE3 was developed and deployed in a private Openstack (Pike) cloud running 24 cores at 2.2 GHz and 256GB of RAM in compute nodes (note that the reused infrastructure services, represented in dashed boxes in Figure 1, run in the controller node). Small virtual machines are chosen to run the simulation service, with a single virtual core and 2GB of RAM. It is interesting to explore if there are benefits in running more than one task at a time in each of them. Note that the eventual gain will the opportunity to exploit context switches in

Table 1 Different scalability polices evaluated to determine the number of simulation instances from the number of pending tasks, parametrized by T (a desired objective in tasks per worker), $minVM$ and $maxVM$ (the minimum and maximum numbers of running simulation instances).

Policiy name	Rule number	Rule type	Condition	Action
P_{none}	#1	Do not scale		$instances := minVM$
P_{full}	#2	Scale up	$\frac{tasks}{instances} > 0$	$instances := maxVM$
	#3	Scale down	$\frac{tasks}{instances} = 0$	$instances := minVM$
P_{cons}	#4	Scale up	$\frac{tasks}{instances} > T$	$instances := instances + 1$
	#5	Scale down	$\frac{tasks}{instances} < \frac{T}{2}$	$instances := instances/2$
P_{aggr}	#6	Scale up	$T < \frac{tasks}{instances} \leq 2T$	$instances := instances + 1$
	#7	Scale up	$2T < \frac{tasks}{instances} \leq 4T$	$instances := 2 \times instances$
	#8	Scale up	$4T < \frac{tasks}{instances} \leq 8T$	$instances := 4 \times instances$
	#9	Scale up	$\frac{tasks}{instances} > 8T$	$instances := 8 \times instances$
	#10	Scale down	$\frac{tasks}{instances} < \frac{T}{2}$	$instances := instances/2$
Common rules	#11	Upper bound	$instances > maxVM$	$instances := maxVM$
	#12	Lower bound	$instances < minVM$	$instances := minVM$

the scheduler of the virtual machine, but also on the decisions taken by the hypervisor of the host machine. To evaluate this issue, experiments are proposed in which each single-core worker can take only one ($N_{tasks} = 1$) or or four ($N_{tasks} = 4$) tasks at at time.

In order to evaluate scalability, synthetic workloads (in the sense that all requests are performed automatically by a script, but running the same simulated models that students use in the course) are defined. To evaluate the impact of increasing workload, the performance is evaluated for parameter sweeps resulting in 50, 100, 500, 1,000, 5,000 and 10,000 individual simulations. Each of this individual simulation has a similar duration of about 4 seconds (in the virtual hardware of our cloud, counting only ns-3 time). To study the effect of the duration of individual simulation tasks, we have also defined a new ns-3 model that takes, for each individual simulation, around 1 second to complete. We will thus speak of *long* and *short* individual simulations when reporting the results.

For the scalability policy parameters, we choose to use $minVM = 1$ and $maxVM = 30$ to evaluate an infrastructure that would have a minimal cost when not used while bounded when fully scaled. In our private cloud deployment the upper bound is also motivated by the available hardware, since running too close to the processor overcommit ratio with computationally intensive loads will make the hypervisor to incur in significant performance losses. The scalability threshold is set to $T = 10$ simulations per instance. If scalability was achieved through tracking rules [2] this would mean that each worker would have a pending workload accounting for about 40 seconds (plus distribution overhead), which ideally would result into an excellent response time, if scalability overheads were negligible. In fact, in our educational context, completing the parameter sweeps in several minutes is indeed a fine result, especially considering that they can take some hours in the average computers available in educational laboratories, which precludes teachers to propose parameter sweep studies to the students. The impact of threshold T, however, should be discussed considering both the fact that Openstack only supports *simple* rules and there is an upper bound on the amount of simulation instances.

For comparison, we also evaluated a typical desktop computer available in our teaching laboratory, running a quad-core Intel(R) Core (TM) i5-2400 at 3,10 GHz with 4GB of RAM. The proposed baseline task consists in the script used by the students in a real course, that simply launches the simulations sequentially and thus no overheads are involved. All the experiments were carried with no other heavy processes running, so competing workload can be neglected. Care should be taken when contrasting the two systems since each virtual core used by DNSE3 services is in fact running (a fraction of time) on a physical core with lower clock rate than the desktop used for comparison.

Following [28], experiments are repeated 3 times to account for the fact that response times are not deter-

ministic (some issues cannot be controlled by DNSE3, like access to input/output devices or hypervisor scheduling decisions, for example). Nevertheless, in all cases variation in results was found to be very low.

5.3 Results

Before evaluating scalability, it is interesting to estimate the overheads incurred by the DNSE3. There are multiple operations beyond actually running the simulations which add up time (i.e. the orchestration service publishes one task in the queue service and uploads input files into the storage service, then some simulation service picks up the task, it downloads files from the storage service, then calls ns-3, and finally uploads results to the storage service and notifies the queue service of the completion of the task; when all individual simulations are complete the orchestration service aggregates the results of the parameter sweep). Therefore, we performed an experiment in which we did not allow DNSE3 to scale (policy P_{none}) running parameter sweeps of different size with *long* tasks (i.e. ns-3 computation-only time of each simulation is around 4 seconds, using the same simulation model tested by the students). The results reported in Table 2 show that DNSE3 in average completes one simulation task in 4.4 seconds (0.4 more than the time required by ns-3). It can also be observed that this distribution overhead is independent of the size of the parameter sweep, suggesting the impact of the orchestration service can be neglected.

Besides, it is worth considering the benefits in parallelizing tasks within small virtual machines. In order to evaluate the influence of N_{tasks} alone, we repeated the experiment executing 4 tasks in parallel in each virtual machine. Results are also shown in Table2, confirming that running single simulations involve some stall time that can be exploited by the scheduler of the virtual machine, and that (in the evaluated conditions) this is not affected by decisions in the host machine. Moreover, this performance gain comes at no cost, since the committed resources are the same. Therefore, for the rest of the experiments we will maintain $N_{tasks} = 4$.

With these parameters ($minVM = 1$, $maxVM = 30$, $N_{tasks} = 4$) several experiments were carried out to evaluate the performance of DNSE3 using different scalability policies to compute parameter sweep jobs from 50 to 10,000 individual *long* simulation tasks. The mean response times are shown in Table 3, along with the time used in a typical laboratory computer (note that the latter does not exploit high level parallelism but has a faster CPU). To facilitate comparison, Figure 2 depicts the speedup of DNSE3 with respect to the laboratory computer in each of the experiments.

Table 2 Mean response times (in seconds) for DNSE3 without scalability, when the simulation service takes $N_{tasks} = 1$ and $N_{tasks} = 4$ tasks at a time, in order to complete parameter sweeps of *long* individual simulations.

	Number of simulations					
N_{tasks}	**50**	**100**	**500**	**1,000**	**5,000**	**10,000**
1	222	457	2,261	4,611	22,079	44,055
4	92	179	822	1,577	7,750	14,386

Table 3 Mean response times (in seconds) for DNSE3 with different scalability policies and the laboratory machine used as baseline, in order to complete parameter sweeps of *long* individual simulations.

	Number of simulations					
Policy	**50**	**100**	**500**	**1,000**	**5,000**	**10,000**
Baseline	88	177	889	1,809	9,188	17,443
P_{none}	92	179	822	1,577	7,750	14,386
P_{all}	89	156	238	273	552	874
P_{cons}	83	157	364	505	1,060	1,522
P_{aggr}	83	154	329	414	689	1,025

Table 4 Mean variable costs (in cents of USD) for DNSE3 with different scalability policies, when used to run parameter sweeps of *long* individual simulations, calculated using AWS billing scheme and prices.

	Number of simulations					
Policy	**50**	**100**	**500**	**1,000**	**5,000**	**10,000**
P_{none}	0	0	0	0	0	0
P_{all}	3.66	3.63	4.84	6.06	11.29	18.13
P_{cons}	0.04	0.19	1.04	1.90	6.92	13.46
P_{aggr}	0.04	0.57	5.84	7.05	13.10	20.34

Besides performance results, cost must also be evaluated. Since our experiments where run in a private cloud, we made an estimation of the cost in a public cloud by measuring the total number of seconds virtual machines where up in each of the experiments, and then calculating its cost according to AWS billing scheme and prices (EU availability zone served from Ireland, April 2018). It must be taken into account that DNSE3 has a permanent infrastructure requiring 6 virtual cores (the orchestration and queue services use 2 cores each, while the report service uses 1; $minVM = 1$ core is also needed for the always-running simulation service), that account for \$0.15/hour. Besides, virtual machines running simulation services can be launched and stopped by the scalability service as needed, resulting into a variable cost that differs for each experiment. These costs are reported in Table 4.

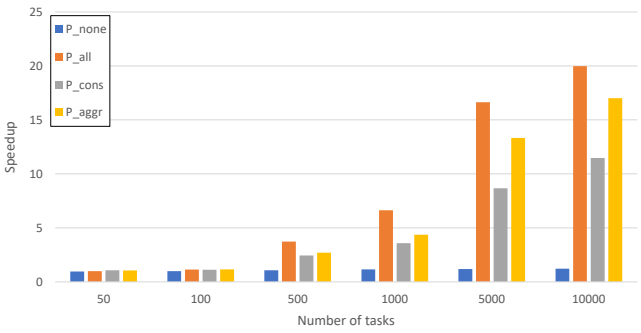

Fig. 2 Speedup in performance (number of times faster) achieved by DNSE3 with different scalability policies used to run parameter sweeps of *long* individual simulations, when compared to the baseline machine.

Results show in the first place that, expectedly, there are not performance gains when scalability is not allowed (policy P_{none}). The fact that the only running simulation service can take $N_{tasks} = 4$ parallel tasks compensates roughly for the overhead of DNSE3 distributed architecture. On the other hand, deploying $maxVM = 30$ simulation services as soon as one task is published in the queue (policy P_{all}) allows completing the longest parameter sweeps very quickly, reaching a speedup of 20 times faster.

Figure 3a shows the evolution of the number of tasks in the queue along with the number of workers known by Openstack's Heat (i.e. the number of instances running *or* requested and still warming up). The figure shows the impact of the delays of the different steps required to apply scalability. Note that DNSE3 controls how often the queue service publishes new measurements of the queue occupation (we set up to do it every two seconds), but these measurements must be published by the monitoring services (Openstack's Ceilometer), and this happens every minute, and then Aodh, Openstack's telemetry service, reads this metric and triggers and alarm that reaches Heat, but this again will happen only every minute. Therefore, Heat will evaluate the scalability policy two minutes after DNSE3 queue service published the occupation of the queue. As a result, Heat will ask Nova (Openstack's compute service) to launch new instances, but this will take some more time. In the Figure, the queue service registers 10,000 tasks at $t = 13$ seconds, and 55 seconds later ($t = 68$) Heat requests 29 new instances. Then the new instances must warm up. It can be observed how the termination of tasks accelerates until a point in which it acquires a new slope, meaning all instances are working at $t = 220$ seconds, 152 seconds after the scalability policy was evaluated.

Despite the performance improvement achieved by policy P_{cons} in large parameter sweeps, for smaller jobs the newly launched simulation instances arrive too late (as expected from the above discussion), and do not help to reduce response times, while they will nevertheless have an impact in cost. Therefore, it seems advisable to explore policies that scale in proportion to the amount of pending work. Policy P_{cons} is probably the simplest, yet most used set of scalability rules expressed in terms of tasks pending in a task pool. It recruits new resources conservatively, achieving performance gains for short parameter sweeps similar to P_{all}, but without overcommitting resources. Nevertheless, for large jobs of 1,000 or more tasks, its provisioning of new resources is too slow. Figure 3b shows how the delay between the two consecutive evaluations of the scalability rules (forced by the fact that previous actions must have been completed) clearly worsens this effect. Alternatively, policy P_{aggr} is more proportional in the request for new resources, achieving similar performance to P_{all} and P_{cons} for smaller parameter sweeps, with a cost similar to the latter. For larger jobs, it improves the performance of P_{cons}, but with a cost even higher than P_{all}, mainly due to the fact that it scales down less quickly. Figure 3c illustrates how new instances are recruited at a faster pace when there are many tasks left in the queue, but again the guard interval between scalability actions penalize somehow the performance. Interestingly, Figure 3b also shows how the delay between metric publication and scalability evaluation can cause one last but unnecessary scalability step, that recruits a new instance when it is no longer necessary.

Another relevant issue to take into account is the duration of the individual task, which affects the "amount" of work in the task pool (e.g. 5,000 *short* tasks may account for the same computation as 1,000 *long* tasks). To explore the effect in DNSE3 performance, and its relationship to the scalability policies we repeated the experiments with parameter sweeps consisting of individual simulations demanding approximating four times less computation (i.e. ns-3 time only is around 1 second). The results are shown in table 5 and Figure 4, where it can be seen that the overhead of distribution in DNSE3 is reducing the speedups achieved in previous experiments (i.e. while for the regular laboratory machine response times are reducing roughly four times, for DNSE3 they are at most halved). Besides, for small parameter sweeps the chosen policy does not make a great difference in performance (though it will in cost, see Table 6), because when the first additional simulation instances are ready, the task queue will have emptied much more than in previous experiments, and thus these instances will have hardly any work to do. Nevertheless, DNSE3 still benefits from parallelization for large parameter sweeps, with bigger speedups for more

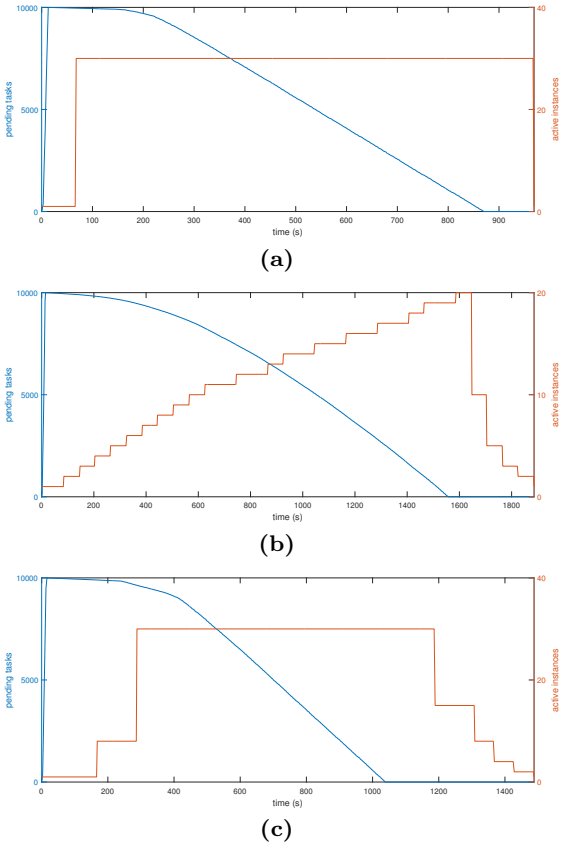

Fig. 3 Evolution of the number of simulation instances and the number of pending tasks when a parameter sweep of 10,000 *long* tasks is requested, using policies **(a)** P_{all}, **(a)** P_{cons} and **(a)** P_{aggr}. Notice that the time axis is different for each of the plots.

Table 5 Mean response times (in seconds) for DNSE3 with different scalability policies and the laboratory machine used as baseline, in order to complete parameter sweeps of *short* individual simulations.

Policy	Number of simulations					
	50	**100**	**500**	**1,000**	**5,000**	**10,000**
Baseline	26	54	219	400	2,088	3,718
P_{none}	47	91	434	850	4,203	8,392
P_{all}	50	90	216	245	417	596
P_{cons}	50	92	271	375	802	1,165
P_{aggr}	52	93	240	291	532	765

aggressive policies, though at a significant increase in cost in the case of P_{aggr}.

5.4 Discussion

From the end user perspective, the results introduced above show how to leverage the computational cloud in the educational realm beyond a range of Software

Table 6 Mean variable costs (in cents of USD) for DNSE3 with different scalability policies, when used to run parameter sweeps of *short* individual simulations, calculated using AWS billing scheme and prices.

Policy	Number of simulations					
	50	**100**	**500**	**1,000**	**5,000**	**10,000**
P_{none}	0	0	0	0	0	0
P_{all}	2.81	3.64	4.83	5.26	8.47	11.67
P_{cons}	0.04	0.04	0.54	1.05	4.29	7.61
P_{aggr}	0.04	0.08	5.84	5.84	9.87	14.28

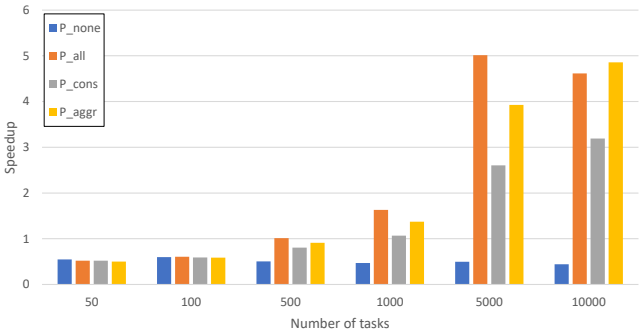

Fig. 4 Speedup in performance (number of times faster) achieved by DNSE3 with different scalability policies used to run parameter sweeps of *short* individual simulations, when compared to the baseline machine.

as a Service generic applications commonly used [22, 44]. Considering that a parameter sweep consisting of 10,000 *long* simulations can take up to five hours in the reference laboratory machine, completing it in 15-20 minutes means a significant advance, that renders them feasible in laboratory classes, so that teachers can exploit the pedagogical benefits of this kind of experiments. Moreover, thanks to the pay per use model of the cloud, it does so at a marginal cost, that can be easily assumed by educational institutions. In addition, the system is not tight to a specific simulation model, and any ns-3 supported model can be run. The trade off between performance and cost could be further explored, as for example done by [39], that combines *on demand* and *spot* instances and introduce cost factors in the scalability policies, though this research is beyond the scope of this paper.

Despite the benefits for the end user, some technical issues can also be discussed. DNSE3 achieves performance gains by parallelization but suffers from several overheads and penalizations that should not be overlooked. For example, the orchestration service publishes in the queue service each of the tasks of a parameter sweep by a POST call, instead of making a bulk publication, because the latter is not strictly RESTful. For the same reason, the simulation service asks first for the next job in the queue (GET) and then takes it (PATH).

These decisions could be revised to favor performance over design, by reducing distribution overheads. Other technological choices to improve performance do not require to change DNSE3. For example, for short parameter sweeps scaling up is typically useless, because of the instance warm up time (note, however, that this may depend on the particular cloud computing infrastructure employed). A potential approach to compensate this fact would be to have more simulation instances permanently running (i.e. to use a higher $minVM$ in scalability rule #12), though fixed costs will also be higher. Nevertheless, if the usage by students can be anticipated (e.g. because laboratory hours are known beforehand) this approach can be followed with reasonable costs.

In addition, though the objective of this paper is not to propose the best possible scalability policy, some findings can be useful to this aim. First, from the observation of Figure 3, it can be noticed that the metric published by the queue service is evaluated by the scalability service with considerable delay. A potential improvement would be to evaluate the rules in Table 1 with the expected number of *current* tasks, i.e. to subtract from the published value the amount of tasks that are expected to have been completed since then. This would imply, on the one hand, modeling the time between metrics publication and scalability evaluation, which can be done easily given the infrastructure configuration parameters. On the other, the average time demanded by each task should also be estimated, though this is not always easy, since simulation models can be eventually developed by students, and therefore the duration of individual tasks cannot be anticipated for certain values of the parameters to sweep.

Figure 3 also shows that the warm up time of instances is not negligible. Modifying the rules to account for the tasks that can be completed by already running instances while additional ones are being launched could also help avoiding provisioning unnecessary resources. Interestingly, by observing that more aggressive scalability policies (P_{all} or P_{aggr}) do not perform any better or can even be worse than more conservative ones, we found out that, at least in our limited private cloud infrastructure, average warm up times are higher when the scalability policies launch more instances at a time. To illustrate this, we varied $maxVM$ in scalability rule #2 and measured the time from the evaluation of the rule by Openstack's Heat to the point where the new simulation services are running. The results are shown in Figure 5, confirming this observation. Therefore, determining the right amount of new instances to start is not only a trade off between cost and performance, as scaling too much can also have a performance penalization in the short term (nevertheless, in a public cloud infrastructure this effect will likely be unperceivable).

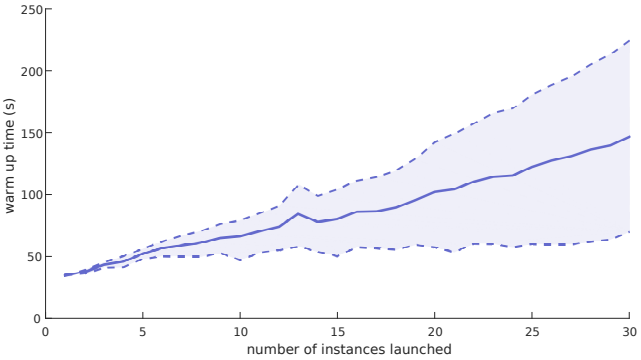

Fig. 5 Warm up times for instances running the simulation service in the experimental Openstack private infrastructure, depending on the number of new instances requested at a time. The dashed lines represent the times for the fastest and slowest instance to warm up, while the solid like shows the average.

Another relevant issue concerns the decisions on when and how to scale down. Scaling down slowly can impose additional costs, as seen in the results of policy P_{aggr} (as compared to P_{all}), though it could be beneficial if more workload is expected in a short term (both to improve performance and reduce the cost per parameter sweep). A prediction model based on the activity of the students and other contextual information could be derived for this purpose.

A final issue to discuss is how to determine threshold T in rules of Table 1. Ideally, it could be chosen to bound the total response time (i.e. if for *long* tasks, each task will take in average 1.4 seconds with only one worker, and the number of workers is sized such that the queue contains $T = 10$ tasks for each, then response time would be around 14 seconds). This is far from real, for several reasons: delays in evaluating scalability; warm up times; the degree of aggressiveness in which resources are provisioned (i.e. how the *simple* rules are somehow achieving the effect of AWS *tracking* rules); the upper bound on the number of instances forced by rule #11 in Table 1 (which is reasonable due to either hardware or cost restrictions, but hinders actually achieving T pending tasks per worker in large workloads); the fact that in-worker parallelization gains depend on decisions taken by the hypervisor (and in a private cloud with limited resources will likely decrease near the overcommit ratio); and, once again, because the duration of an individual task may be unknown. In addition, if rules aim at achieving certain response times for each parameter sweep requested, the aggregation of workload should also be taken into account. Deriving a model that estimates the impact of each of

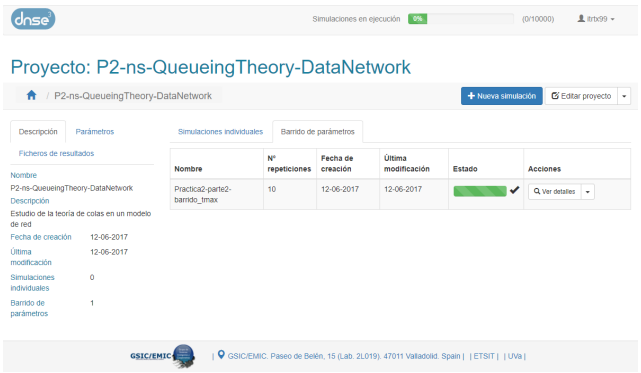

Fig. 6 Snapshot of the web client graphical interface, when the user has demanded to run a parameter sweep and it has finished.

these items, and proposing a scalability policy that attempts to bound job response times is an interesting research, but goes beyond the scope of this paper. Note, additionally, that several of these issues would still need research even if using AWS tracking rules (i.e. even if the auto-scalability mechanism takes the best decisions to guarantee T tasks per worker, it is not simple to select a value of T that serves to attain a certain response time for each parameter sweep, even without guarantees).

6 User interface and usability evaluation

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 6. 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 [8], in the search of compatibility with different browsers and devices, particularly smartphones and tables, which are popular among students.

Though it is reasonable to expect that students will prefer a GUI over using scripts, usability is never guaranteed and must be assessed to confirm this hypothesis or detect potential defects in the interface design that hinder DNSE3 user adoption. Therefore, we requested students already enrolled in a Computer Networks course in which they used ns-3 to run parameter sweeps (creating scripts to generate the individual simulations, compile the model and call ns-3) to participate in a experiment to evaluate DNSE3. Ten students volunteered to join 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 ns-3) and that of DNSE3 were surveyed.

Following industry standards, the System Usability Scale (SUS) [11] was employed, in the version modified by [5], 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 [31,32]. It consists of ten questions that yield an overall score between 0 and 100. [5] suggests not to report answers to each individual questions, since they are highly correlated, and inform just the SUS instead, and proposes a rule of thumb to translate this score into a literal adjective describing the usability of the system. In addition, students were asked their Likelihood To Recommend (LTR) the system to a user with such a need (i.e. another student taking the course), which can serve as a means for validation, since it is strongly correlated with SUS [30].

Overall, DNSE3 obtained a SUS score of 91.75 ("excellent" according to [5]) while the alternative software obtained only 47.75 ("poor"). The sample size does not allow to generalize these results (even if [12] 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.

7 Conclusions and future work

Though parameter sweep simulations can significantly aid to understand the behavior of protocols in a computer network, their use in education is not extended due to their significant computational load. Though cloud computing usages in education have so far been

mostly limited to Software as a Service generic tools, this particular problem features high level parallelism that could be exploited, along with the automatic scalability and pay-per-use features of cloud computing to reduce response times while keeping costs low. 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 work stealing scheduling approach with a single task queue exposing simulation tasks with a two-level ordering, aimed at a fair share of resources among users, and a number of independent simulation workers taking tasks from that queue that is determined dynamically after a given scalability policy.

The evaluation of DNSE3 has shown, on the one hand, that it offers a feasible solution to reach reasonable response times for academic assignments to be carried out during laboratory hours, and it does so at very small costs. The application can be used to sweep any ns-3 compatible simulation model, and scalability policies can be chosen to trade off performance and cost. Together with good acceptance of the user interface by student, DNSE3 opens the way to using parameter sweep simulation for educational purposes in Computer Network courses. Future work along this line aims at evaluating the application in several real courses and model the distribution of parameter sweeps requested by the students, in order to use this probabilistic model to tune up the cost/performance ratio.

On the other hand, the paper has discussed some issues in designing the policies that control the self-scalability mechanisms. For applications in which the the relevant performance metric is the response time of a job (a parameter sweep) consisting of many tasks (individual simulations), the number of tasks pending per worker could be used as a scalability metric, and hence dimension the number of workers towards this goal. However, achieving the ideal response time is hindered by several issues, like delays from metric publication to scalability decisions, the warm up times, the aggressiveness of scalability policies, and whether the time taken by individual tasks can be estimated beforehand and considered in the definition of the desired tasks per worker thresholds of scalability rules. Our future work plans to explore all these issues by modeling the estimated computation time by each individual simulation and defining scalability accordingly.

References

1. Amazon Web Services, Inc.: Amazon Web Services. URL https://aws.amazon.com/
2. Amazon Web Services, Inc.: What Is Amazon EC2 Auto Scaling? URL https://docs.aws.amazon.com/autoscaling/ec2/userguide/
3. The Apache Software Foundation: Apache CloudStack Open Source Cloud Computing. URL https://cloudstack.apache.org/
4. Arora, N.S., Blumofe, R.D., Plaxton, C.G.: Thread scheduling for multiprogrammed multiprocessors. In: ACM Symposium on Parallel Algorithms and Architectures, pp. 119–129 (1998). DOI 10.1145/277651.277678
5. Bangor, A., Kortum, P.T., Miller, J.T.: An empirical evaluation of the system usability scale. International Journal of Human & Computer Interaction **24**(6), 574–594 (2008). DOI 10.1080/10447310802205776
6. Behrisch, M., Bieker, L., Erdmann, J., Krajzewicz, D.: SUMO – simulation of urban mobility. In: International Conference on Advances in System Simulation, pp. 63–68 (2011)
7. Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. J. ACM **46**(5), 720–748 (1999). DOI 10.1145/324133.324234
8. Bootstrap: an open source toolkit for developing with HTML, CSS, and JS. URL http://getbootstrap.com/
9. Bote-Lorenzo, M.L., Asensio-Pérez, J.I., Gómez-Sánchez, E., Vega-Gorgojo, G., Alario-Hoyos, C.: A grid service-based distributed network simulation environment for computer networks education. Computer Applications in Engineering Education **20**(4), 654–665 (2012). DOI 10.1002/cae.20435
10. Bragard, Q., Ventresque, A., Murphy, L.: Self-balancing decentralized distributed platform for urban traffic simulation. IEEE Transactions on Intelligent Transportation Systems **18**(5), 1190–1197 (2017). DOI 10.1109/TITS.2016.2603171
11. Brooke, J.: SUS: A quick and dirty usability scale. In: P.W. Jordan, B. Thomas, I.L. McClelland, B. Weerdmeester (eds.) Usability evaluation in industry, pp. 189–194. Taylor & Francis, London, UK (1996)
12. Brooke, J.: SUS: A retrospective. J. Usability Studies **8**(2), 29–40 (2013)
13. Caglar, F., Shekhar, S., Gokhale, A., Basu, S., Rafi, T., Kinnebrew, J., Biswas, G.: Cloud-hosted simulation-as-a-service for high school {STEM} education. Simulation Modelling Practice and Theory (2015). DOI 10.1016/j.simpat.2015.06.006
14. Calcavecchia, N.M., Caprarescu, B.A., Di Nitto, E., Dubois, D.J., Petcu, D.: Depas: a decentralized probabilistic algorithm for auto-scaling. Computing **94**(8), 701–730 (2012). DOI 10.1007/s00607-012-0198-8. URL https://doi.org/10.1007/s00607-012-0198-8
15. Cao, Y., Jin, X., Li, Z.: A distributed simulation system and its application. Simulation Modelling Practice and Theory **15**(1), 21 – 31 (2007). DOI 10.1016/j.simpat.2006.09.010
16. Evangelidis, A., Parker, D., Bahsoon, R.: Performance modelling and verification of cloud-based auto-scaling policies. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 355–364 (2017). DOI 10.1109/CCGRID.2017.39
17. Fielding, R.T.: Architectural syles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine, USA (2000)
18. Foster, I.: Globus toolkit version 4: Software for service-oriented systems. Journal of Computer Science and Technology **21**(4), 513 (2006). DOI 10.1007/s11390-006-0513-y. URL https://doi.org/10.1007/s11390-006-0513-y

19. Fujimoto, R.M.: Research challenges in parallel and distributed simulation. ACM Transactions on Modeling and Computer Simulation (2016). DOI 10.1145/2866577
20. Fujimoto, R.M., Malik Fujimoto, R.M., Malik, A.W.: Parallel and distributed simulation in the cloud. SCS Modeling and Simulation Magazine **1**(3) (2010)
21. Ghanbari, H., Simmons, B., Litoiu, M., Iszlai, G.: Exploring alternative approaches to implement an elasticity policy. In: 2011 IEEE 4th International Conference on Cloud Computing, pp. 716–723 (2011). DOI 10.1109/CLOUD.2011.101
22. González-Martínez, J.A., Bote-Lorenzo, M.L., Gómez-Sánchez, E., Cano-Parra, R.: Cloud computing and education: A state-of-the-art survey. Computers & Education **80**, 132 – 151 (2015). DOI 10.1016/j.compedu.2014.08.017
23. Google LLC: Google App Engine. URL https://cloud.google.com/appengine
24. Hasan, M.Z., Magana, E., Clemm, A., Tucker, L., Gudreddi, S.L.D.: Integrated and autonomic cloud resource scaling. In: 2012 IEEE Network Operations and Management Symposium, pp. 1327–1334 (2012). DOI 10.1109/NOMS.2012.6212070
25. Huang, C.S., Tsai, M.F., Huang, P.H., Su, L.D., Lee, K.S.: Distributed asteroid discovery system for large astronomical data. Journal of Network and Computer Applications **93**(Supplement C), 27 – 37 (2017). DOI 10.1016/j.jnca.2017.03.013
26. Hüning, C., Adebahr, M., Thiel-Clemen, T., Dalski, J., Lenfers, U., Grundmann, L.: Modeling & simulation as a service with the massive multi-agent system MARS. In: Agent-Directed Simulation Symposium, pp. 1–8 (2016)
27. Indhumathi, V., Nasira, G.M.: Service oriented architecture for load balancing with fault tolerant in grid computing. In: IEEE International Conference on Advances in Computer Applications (ICACA), pp. 313–317 (2016). DOI 10.1109/ICACA.2016.7887972
28. Jain, R.: The art of computer systems performance analysis: Techniques for experimental design, measurement, simulations and modelling. John Wiley & Sons, New York, NY, USA (1991)
29. Law, A.M., Kelton, W.D.: Simulation modeling and analysis. McGraw-Hill, New York, NY, USA (1991)
30. Lewis, J.R.: Usability: Lessons learned... and yet to be learned. International Journal of Human & Computer Interaction **30**(9), 663–684 (2014). DOI 10.1080/10447318.2014.930311
31. Lin, H.C.K., Chen, M.C., Chang, C.K.: Assessing the effectiveness of learning solid geometry by using an augmented reality-assisted learning system. Interactive Learning Environments **23**(6), 799–810 (2015). DOI 10.1080/10494820.2013.817435
32. Martin-Gonzalez, A., Chi-Poot, A., Uc-Cetina, V.: Usability evaluation of an augmented reality system for teaching euclidean vectors. Innovations in Education and Teaching International **53**(6), 627–636 (2016). DOI 10.1080/14703297.2015.1108856
33. Microsoft: Microsoft Azure. URL https://azure.microsoft.com/
34. The Network Simulator - ns-2. URL http://www.isi.edu/nsnam/ns/
35. The Network Simulator - ns-3. URL https://www.nsnam.org/
36. OASIS: OASIS Web Services Resource Framework (WSRF). URL https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
37. OpenStack: OpenStack Open Source Cloud Computing Software. URL https://www.openstack.org/
38. Papadopoulos, C., Heidemann, J.: Using ns in the classroom and lab. In: ACM SIGCOMM Workshop on Computer Networking, pp. 45–46. Pittsburgh, PA, USA (2002)
39. Qu, C., Calheiros, R.N., Buyya, R.: A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. Journal of Network and Computer Applications **65**(Supplement C), 167 – 180 (2016). DOI 10.1016/j.jnca.2016.03.001
40. Qun, Z.A., Jun, W.: Application of ns2 in education of computer networks. In: IEEE International Conference on Advanced Computer Theory and Engineering, pp. 368–372 (2008). DOI 10.1109/ICACTE.2008.89
41. Ravindhren, V.G., Ravimaran, S.: Ccma—cloud critical metric assessment framework for scientific computing. Cluster Computing (2017). DOI 10.1007/s10586-017-1384-4. URL https://doi.org/10.1007/s10586-017-1384-4
42. Restlet, Inc.: Restlet Framework. URL https://restlet.com/open-source/
43. Roy, N., Dubey, A., Gokhale, A.: Efficient autoscaling in the cloud using predictive models for workload forecasting. In: 2011 IEEE 4th International Conference on Cloud Computing, pp. 500–507 (2011). DOI 10.1109/CLOUD.2011.42
44. Tashkandi, A.N., Al-Jabri, I.M.: Cloud computing adoption by higher education institutions in saudi arabia: an exploratory study. Cluster Computing **18**(4), 1527–1537 (2015). DOI 10.1007/s10586-015-0490-4. URL https://doi.org/10.1007/s10586-015-0490-4
45. Vaquero, L.M., Rodero-Merino, L., Buyya, R.: Dynamically scaling applications in the cloud. SIGCOMM Comput. Commun. Rev. **41**(1), 45–52 (2011). DOI 10.1145/1925861.1925869. URL http://doi.acm.org/10.1145/1925861.1925869
46. Vaquero, L.M., Rodero-Merino, L., Buyya, R.: Cloud scalability: building the millennium falcon. Concurrency and Computation: Practice and Experience **25**(12), 1623–1627 (2013). DOI 10.1002/cpe.3008
47. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: Towards a cloud definition. SIGCOMM Comput. Commun. Rev. **39**(1), 50–55 (2008). DOI 10.1145/1496091.1496100
48. Vinoski, S.: REST eye for the SOA guy. IEEE Internet Computing **11**(1), 82–84 (2007). DOI 10.1109/MIC.2007.22
49. Wang, A., Jiang, W.: Teaching wireless local area network course based on ns-3. In: International Symposium on Computer Network and Multimedia Technology, pp. 1–4 (2009). DOI 10.1109/CNMT.2009.5374600
50. Wang, S.Y., Lin, C.C., Tzeng, Y.S., Huang, W.G., Ho, T.W.: Exploiting event-level parallelism for parallel network simulation on multicore systems. IEEE Transactions on Parallel and Distributed Systems **23**(4), 659–667 (2012). DOI 10.1109/TPDS.2011.215
51. Weingartner, E., vom Lehn, H., Wehrle, K.: A performance comparison of recent network simulators. In: IEEE International Conference on Communications, pp. 1–5 (2009). DOI 10.1109/ICC.2009.5198657
52. Zehe, D., Knoll, A., Cai, W., Aydt, H.: Semsim cloud service: Large-scale urban systems simulation in the cloud. Simulation Modelling Practice and Theory **58**, 157 – 171 (2015). DOI 10.1016/j.simpat.2015.05.005
53. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications **1**(1), 7–

18 (2010). DOI 10.1007/s13174-010-0007-6. URL http://dx.doi.org/10.1007/s13174-010-0007-6

54. Zhou, X., Tian, H.: Comparison on network simulation techniques. In: International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), pp. 313–316 (2016). DOI 10.1109/PDCAT.2016.073

Photograph of Sergio Serrano-Iglesias

Phtograph of Eduardo Gómez-Sánchez

Phtograph of Miguel L. Bote Lorenzo

Phtograph of Juan I. Asensio-Pérez

Phtograph of Manuel Rodríguez-Cayetano

Short bios of the authors of the manuscript entitled "A self-scalable distributed network simulation environment based on cloud computing" submitted to *Cluster Computing: The Journal of Networks, Software Tools and Applications*.

Sergio Serrano-Iglesias received his BSc and MSc in Telecommunications Engineering from the University of Valladolid, Spain, in 2016 and 2017 respectively. He is currently a PhD candidate in the GSIC-EMIC Research Group at the University of Valladolid. His main research interests include cloud computing and its application for the support of learning.

Eduardo Gómez-Sánchez received his PhD degree in Telecommunications Engineering in 2001. He is currently an associate professor at the University of Valladolid. His main research interest is the design and evaluation of software distributed systems and their application to enhance collaborative learning.

Miguel L. Bote-Lorenzo received his MSc and PhD degrees in Telecommunications Engineering in 2001 and 2005 respectively. He is currently an associate professor at the Department of Signal Theory, Communications and Telematics Engineering, University of Valladolid. His main research interests include distributed systems, machine learning and their application in the context of computer networks and technology enhanced learning.

Juan I. Asensio-Pérez received the Ph.D. degree in Telecommunications Engineering from the University of Valladolid, Spain, in 2000 where he is currently associate professor. His research expertise include the development and management of distributed systems for supporting physical and virtual educational contexts.

Manuel Rodríguez-Cayetano received his MSc and PhD degrees in Telecommunications Engineering in 1993 and 1999 respectively. He is currently an associate professor at the Department of Signal Theory, Communications and Telematics Engineering, University of Valladolid. His main research interests include software development for parallel and distributed systems.