Annals of Telecommunications manuscript No.
(will be inserted by the editor)

A Simulation as a Service Cloud Middleware

Shashank Shekhar, Hamzah Abdelaziz, Michael Walker, Faruk Caglar, Aniruddha Gokhale, Xenofon Koutsoukos

Received: date / Accepted: date

Abstract Recent advances in cloud computing have opened up new avenues for individuals and organizations with limited resources to obtain answers to problems that hiterto required expensive and computationally-intensive resources. One such problem is determining the right thermostat settings for different rooms of a house based on a tolerance range specified by a user such that energy consumption in the house can be maximally reduced while still offering comfortable temperatures in the house. Since models of these systems are often stochastic, the simulations require stochastic model checking where a large number of complex simulations of the models must be executed in parallel and the results aggregated to ascertain if the outcomes lie within specified confidence intervals. A cloud offers an attractive platform for these use cases. To support these capabilities, this paper describes a Cloud-based Simulation-as-a-Service (SIMaaS) middleware. We demonstrate how lightweight solutions using Linux containers (e.g., Docker) are better suited to support such services instead of heavyweight hypervisor-based solutions, which are shown to incur substantial overhead in provisioning virtual machines on-demand.

Keywords Cloud Computing · Middleware · Linux Container · Simulation-as-a-Service

Shashank Shekhar, Hamzah Abdelaziz, Michael Walker, Faruk Caglar, Aniruddha Gokhale and Xenofon Koutsoukos
Department of Electrical Engineering and Computer Science,
Vanderbilt University, Nashville, TN 37235, USA

E-mail: {shashank.shekhar,hamzah.abdelaziz,michael.a.walker.1,faruk.caglar,
a.gokhale,xenonfon.koutsoukos}@vanderbilt.edu

1 Introduction

With the advent of the Internet of Things (IoT) [6] paradigm, which involves the ubiquitous presence of sensors, there is no dearth of gathered data. When coupled with technology advances in mobile computing and edge devices, users are expecting newer and different kinds of services that will help them in their daily lives. For example, users may want to determine appropriate temperature settings for their homes such that their energy consumption and energy bills are kept low yet they have comfortable conditions in their homes. Other examples include estimating traffic congestion in a specific part of a city on a special events day. For all these new services, users expect a sufficiently low response time from the services.

Deploying these services in-house is unrealistic for the users since the models of these systems are complex to develop and stochastic in nature, which require a large number of compute-intensive simulations to obtain outcomes that are within a desired statistical confidence interval. Users cannot be expected to acquire the needed resources in-house. The cloud becomes an attractive option to host such services particularly when hosting high performance and real-time applications in the cloud is gaining traction [22, 4]. Examples include soft real-time applications such as online video streaming (e.g., Netflix hosted in Amazon EC2), gaming (Microsoft's Xbox One and Sony's Playstation Now) and telecommunication management [13].

Insights from prior efforts [11, 17, 18, 19] that focused on deploying parallel discrete event simulations (PDES) [10] in the cloud reveal that performance of the simulation deteriorates as the size of the cluster distributed across the cloud increases. This occurs due primarily to limited bandwidth and the overhead of time synchronization protocols [30] needed on the cloud. Thus, cloud deployment for this category of simulations is still limited.

Despite these insights, we surmise that there is another category of simulations that can benefit from cloud computing. Complex system simulations that require statistical validation or those that compare simulation results under different constraints often need to run repeatedly. Running these simulations sequentially is not a viable option as user expectations in terms of response times have to be met. Hence there is a need for a simulation platform where independent simulation instances can be executed in parallel and the number of such simulations can vary elastically to satisfy specified confidence intervals for the results. A cloud is an attractive platform to host such capabilities, which we have architected in the form of a Simulation-as-a-Service (SIMaaS) [29] middleware for the cloud.

A SIMaaS built on top of traditional cloud infrastructure would likely utilize a virtual machine (VM)-based data center to provide resource sharing. However, in a scenario where real time decisions have to be made based on running large number of multiple, short-duration simulations in parallel, the considerable setup and tear down overhead imposed by VMs is unacceptable. A solution based on maintaining a VM pool that is used by many cloud resource management frameworks such as [16, 8, 32, 12], however, leads to resource wastage and may not be able to cater to sudden increases in service demand.

To address these challenges, we make the following key contributions in this paper:

– We propose a cloud middleware for SIMaaS that applies Linux container [20]-based infrastructure which has low runtime overhead, higher level of resource sharing, and very low setup and tear down costs.
– We show how the middleware intelligently generates different configurations for experimentation, intelligently schedules the simulations on the Linux container-based cloud to minimize cost, and meets deadlines.
– Using a case study, we show the viability of a Linux container-based SIMaaS solution, and illustrate the performance gains of a Linux container-based approach over a hypervisor-based traditional virtualization techniques used in the cloud.

The rest of this paper is organized as follows: Section 2 deals with relevant related work comparing it with our contributions; Section 3 provides a use case and formulates the problem we solve in this research; Section 4 presents the system architecture in detail; Section 5 validates the effectiveness of our middleware; and finally Section 6 presents concluding remarks alluding to lessons learned and opportunities for future work.

2 Related Work

This section presents relevant related work and compares them with our contributions. We provide related work along three facets: simulations hosted in the cloud, cloud frameworks that provide resource management with deadlines, and container-based approaches. These facets of related work are important since realizing SIMaaS requires effective resource management at the cloud infrastructure-level to manage the lifecycle of containers that host and execute the simulation logic such that user-specified deadlines are met.

2.1 Related Work on Cloud-based Simulations

The mJADES [25] effort is closest to our approach in terms of its objective of supporting simulations in the cloud. It is founded on a Java-based architecture and is designed to run multiple concurrent simulations while automatically acquiring resources from an ad hoc federation of cloud providers. DEXSim [9] is a distributed execution framework for replicated simulations that provides two-level parallelism, i.e., at CPU core-level and at system-level. This organization delivers better performance to their system. In contrast, SIMaaS does not provide any such scheme; rather it relies on the OS to make effective use of the multiple cores on the physical server by pinning container processes to cores. The RESTful interoperability simulation environment (RISE) [3] is a cloud middleware that applies RESTful APIs to interface with the simulators and allows their management remotely through Android-based handheld devices. Like RISE, SIMaaS also uses RESTful APIs for clients to interact with our service and for the internal interaction between the containers and the management solution. In contrast to these works, SIMaaS applies an adaptive resource scheduling policy to meet the deadlines based on the current system performance. Also, our solution uses the Linux container-based simulation cloud that is more efficient and more suitable to the kinds of simulations hosted by SIMaaS than the VM-based approach used by these solutions.

2.2 Related Work on Cloud Resource Management

There has been some work in cloud resource management to meet deadlines. Aneka [8] is a cloud platform that supports quality of service (QoS)-aware provisioning and execution of applications in cloud. It supports different programming models, such as bag of tasks, distributed threads, MapReduce, actors and workflows. Our work on SIMaaS uses a resource management algorithm that is a variation of the one used by Aneka in the context of our Linux container-based lightweight virtualization solution. It also provides algorithms to provision hybrid clouds to minimize the cost and meet deadlines. Although SIMaaS does not use hybrid clouds, our future work will consider some of the functionalities from Aneka.

CometCloud [16] is a cloud framework that provides autonomic workflow management by addressing changing computational and QoS requirements. It adapts both application and infrastructure to fulfill its purpose. CLOUDRB [28] is a cloud resource broker that integrates deadline based job scheduling policy with particle swarm optimization-based resource scheduling mechanism to minimize both cost and execution time to meet a user-specified deadline. Zhu et al. [32] employed a rolling-horizon optimization policy to develop energy-aware cloud data center for real-time task scheduling. All these efforts provide scheduling algorithms to meet deadlines on virtual machine-based cloud platform where they maintain a VM pool and scale up or down based on constraints. In contrast to these efforts, our work uses a lightweight virtualization technology based on Linux containers which provides significant performance improvement and mitigates the need to keep a pool of VMs or containers. Nonetheless, our solution also requires container scheduling.

2.3 Related Work using Linux Containers

The Docker [24] open source project that we utilize in our framework automates the deployment of applications via software containers utilizing operating system (OS)-level virtualization. Docker is not an OS-level virtualization solution; rather it uses interchangable execution enviroments such as Linux Containers (LXC) and its own *libcontainer* library to provide Container access and control.

Previous work exists on the creation [23] and benchmark testing [27] of generic Linux-based containers. Similarly, there exists work that use containers as a means to provide isolation and a lightweight replacement to hypervisors in use cases such as high performance computing (HPC) [31], reproducible network experiments [14], and peer-to-peer testing enviroments [7]. The demands and goals of each of these three efforts focuses on a different aspect of the benefit stemming from the use of containers. For HPC, the effort focused more on the lightweight nature of containers versus hypervisors. The peer-to-peer testing work focused on the isolation capabilities of containers whereas the reproducible network experiments paper focused more on the isolation features and the ability to distribute containers as deliverables for others to use in their own testing. Our work leverages or can leverage all these benefits.

3 Motivating Use Case and Problem Statement

We now present a use case scenario belonging to a specific class of systems modeling that we have used in this paper to demonstrate the capabilities of the SIMaaS cloud middleware.

3.1 Systems Modeling Support in SIMaaS

System modeling for simulation is a very rich area that has been used in a wide range of different engineering disciplines. The type of system modeling depends on the nature of the system to be modeled and the level of abstraction needed to be achieved through the simulation.

In this paper we target complex engineering systems which exhibit continuous, discrete, and probabilistic behaviors, known as stochastic hybrid systems (SHS), such as air traffic control systems. The computer model we use to construct a formal representation of a SHS systems and to mathematically analyze and verify it in a computer system is *discrete time stochastic hybrid system (DTSHS)* models [1].

3.2 Use Case: The Multi-room Heating System

For this paper we have used a DTSHS model in a multi-room heating system scenario [5] with its discretized model developed by [2]. The multi-room heating system consists of h rooms and a limited number of heaters n where $n < h$. Each room can have at most one heater at a time. Moreover, each room has its own user setting (i.e., constraints) for temperature setting. However, each room has an exchangeable effect with its adjacent rooms and with the ambience.

Each room heater switches independently of the heater status of other rooms and their temperatures. The system has a hybrid state where the discrete component is the state of individual heater, which can be in ON or OFF state, and the continuous state is the room temperature. A discrete transition function switches the heaters status in each room based on using a typical differential controller which switches the heater on if the room temperature get below a certain threshold xl and switches the heater off if the room temperature exceed a certain threshold xu.

3.3 Problem Statement and Key Requirements

The main challenge for our use case is the limited number of heaters and the need for a control strategy to move a heater between the rooms. The system requirements to evaluate the model are:

– The temperature in each room must always remain above a certain threshold (i.e., user comfort level).
– All rooms share heaters with other rooms (i.e., acquire and relinquish a heater).

We have used one strategy where room i can acquire a heater with a probability p_i if:

– $p_i \propto get_i - x_i$ when $x_i < get_i$.
– $p_i = 0$ when $x_i \geq get_i$.

where get_i, dif_i are control thresholds that are used to determine when room i needs to acquire a heater and when room i can actually retrieve a heater from room j, respectively.

These requirements can be evaluated through model simulations; however, every simulation may yield a different simulation trajectory and results because of the model's stochastic nature. To overcome this problem, we use the *statistical model checking (SMC)* approach based on Bayesian statistics [32, 33]. SMC is a verification method that provides statistical evidence to check whether a stochastic system satisfies a wide range of temporal properties with a certain probability and confidence level or not. The probability that the model satisfies a property can be estimated by running multiple simulation trajectories of the model and dividing the number of satisfied trajectories (i.e., true properties) over the total number of the simulations.

It is precisely for this purpose that the cloud platform is an attractive choice to execute the multiple different simulation trajectories of the stochastic model in parallel, and perform SMC to obtain results within a desired confidence interval. The challenge stems from provisioning these simulation trajectories in the cloud in real-time so that the response times perceived by the user are acceptable. This is the challenge we address through our SIMaaS cloud middleware.

4 SIMaaS Cloud Middleware Architecture

In this work our goal is to support a simulation-as-a-service in the cloud. We expect the users to access our service using traditional web technologies.

Consequently, our system requires a web server to handle user requests. On a per user-basis we require our system to provide elastic resource provisioning such that a sufficient number of simulation runs can be instantiated on-demand for the stochastic model checking and results returned to the user within the deadlines specified by the user. Finally, the system should scale to a large number of users. To address these needs, we have architected the SIMaaS cloud middleware as shown in Figure 1 and described in the rest of this section.

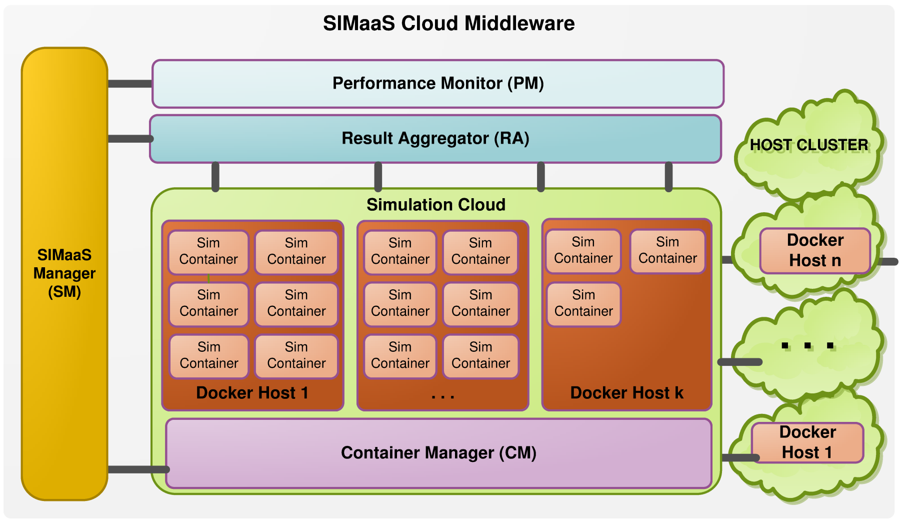

Fig. 1 System Architecture

4.1 System Design and Rationale

We now justify the need for the key artifacts of our design.

4.1.1 Request Handling and Dynamic Resource Provisioning

The central component of the SIMaaS middleware that is responsible for resource provisioning and handling user requests is the SIMaaS Manager (SM). All the coordination and decision making responsibilities lie with this entity. It has a pluggable design that is used to strategize the virtualization approach to be used by the hosted system. Additionally, it also acts as the interface to the user by including a web server. To ensure that the web server does not become a bottleneck, we have used Cherrypy [15] to accept user inputs for simulation and receiving feedback from other SIMaaS components.

SM has a modular architecture and can plug in different resource and scheduling policies, such as time-based, cost-based or resource consumption-based. For this paper we have defined a QoS-based resource provisioning policy shown in Algorithm 1, which calculates the extra number of hosts needed at runtime to meet user-specified deadlines.

```
Input: Host Capacity
forall the User Simulation Requests do
    while Has Pending Simulations do
        Input: Average Execution Time, Remaining Time, Remaining Simulation
               Count, Assigned Capacity, Buffer Time
        Output: Extra Hosts Needed
        Remaining Time = Remaining Time - Buffer Time ;
        Containers per Cycle = (Remaining Simulation Count * Average Execution
        Time) / Remaining Time) ;
        Extra Hosts Needed = (Containers per Cycle - Assigned Capacity) / Host
        Capacity ;
    end
end
```

Algorithm 1: QoS-based Resource Allocation Policy

4.1.2 Lightweight Virtualization Management

A cloud platform typically uses virtualized resources to host user applications. Different forms of virtualization include full virtualization (e.g., KVM), paravirtualization (e.g., Xen) and lightweight containers (e.g., LXC Linux containers). Since full and para virtualization require the entire OS to be booted from scratch whenever a new virtual machine (VM) is needed, this bootup time incurs a delay in availability of new VMs not to mention the cost of the application's initialization time. Hence, SIMaaS uses the lightweight containers, which suffice for our purpose.

The lifecycle of these containers is managed by the Container Manager (CM). The pluggable architecture of SM allows CM to switch between various container providers that can be Linux container or hypervisor-based VM cloud. The Linux container is the default container provider. Specifically, we use Docker [24] container virtualization technology since it provides portable deployment of Linux containers and provides a registry for images to be shared across the hosts with significant performance gains over hypervisor-based approaches. Thus, the CM is responsible for keeping track of the hosts in the cluster and provision the running and tear down of the Docker containers. It downloads and deploys different images from the Docker registry for instantiating different simulations on the cluster hosts.

Our earlier design of the CM leveraged Shipyard [26] for communicating with the Docker hosts, however, due to sluggish performance we observed, we had to implement a custom solution with reduced role of Shipyard. Overcoming the reasons for the sluggish performance and reusing existing artifacts maximally is part of our future investigations.

4.1.3 Cloud Instrumentation Design

Recall that meeting user-specified deadlines is an important goal for SIMaaS. These deadlines must be met in the context of the stochastic model checking that requires multiple simultaneous runs of the stochastic simulation models. Thus, SIMaaS must be cognizant of overall system performance and make dynamic resource management decisions, which requires effective system instrumentation.

Since SIMaaS uses Linux containers, we leveraged the Performance Monitor (PerfMon) package from the JMeter Plugins group of packages on Linux. PerfMon is an open-source Java application which runs as a service on the hosts to be monitored. Since the monitored statistics are required by the Performance Manager (PM) component instead of a visual rendition, we implemented custom software to tap into PerfMon via its TCP/UDP connection capabilities. PerfMon is by no means the only option available but it sufficed our needs.

PerfMon depends on the SIGAR API and uses that for its gathering of system metrics. The metrics available are classified into 8 broad catagories. These catagories include: CPU, Memory, Disk I/O, Network I/O, JMX (Java Management Extensions), TCP, Swap, and Custom executions. We are currently not using the JMX, TCP, or Swap merics, but they are available for use if needed. Each of these catagories have parameters to allow customization of desired returned metrics, e.g., Custom allows for the returning of any custom command line execution. We use this to execute a custom script that returns the process id and container id pairs of each running Docker container. This allows us to monitor each individual container performance precisely.

4.1.4 Result Aggregation

Stochastic model checking requires that results of the multiple simulation runs be aggregated to ascertain if the specified probabilistic property is met or not. To do this, a key component of our middleware is the Result Aggregator (RA). RA receives the simulation results from the Docker containers. It uses ZeroMQ for reliable result delivery and sends feedback to the SM for decision making. Since the aggregation logic and model checking is application-dependent, it is supplied by the user when the service is hosted, and is used when a sufficient number of results are received. We realize that due to the large number of messages received by this system, it can become the bottleneck for the system and we plan to use a load balanced architecture in future that can also provide fault tolerance.

4.2 SIMaaS Workflow and User Interaction

We now describe how a user interacts with SIMaaS and the workflow triggered by a user interaction. A user contacts the SM component of SIMaaS and provides the initial configuration which includes the simulation executables, resource requirement for each simulation and the aggregation logic. An image is generated and deployed on the cloud registry accessible by the container hosts. The aggregation logic is deployed on the Result Aggregator component.

Figure 2 depicts the runtime flow of SIMaaS. After the user input, the SM applies a resource allocation and scheduling policy, and calculates the minimum number of hosts needed. It then contacts the CM and starts the simulation containers. The containers log the result to RA that keeps sending feedback to the SM and performs the aggregation when the desired number of simulation results are received. SM also runs a service to determine if deadlines will be met based on current performance data, and accordingly contacts the CM to acquire additional resources and start simulations.

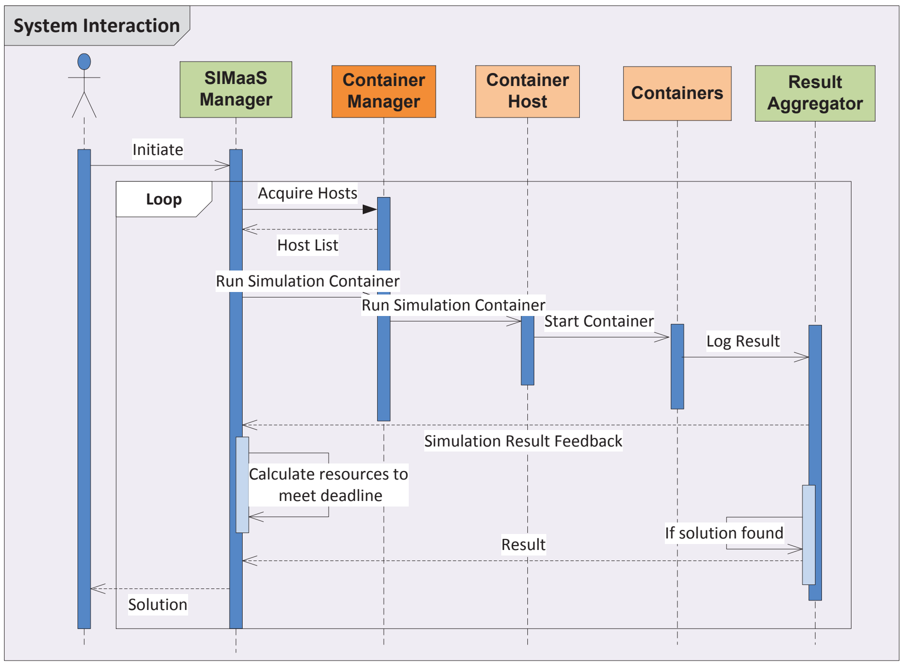

Fig. 2 SIMaaS Interaction Diagram

5 Experimental Validation

This section evaluates the performance properties of the SIMaaS middleware in the context of hosting the use case described in Section 3.2. Since SIMaaS has a pluggable strategy, we also compare the performance differences between using a container and traditional virtualization approaches.

5.1 Experimental Setup

Our setup consists of six nodes each with the configuration defined in Table 1. The same set of machines were used for experiments for both Linux containers and virtual machines. Docker version 1.2.0 was used for Linux container virtualization and QEMU-KVM was used for hypervisor virtualization with QEMU version 2.0.0 and Linux kernel 3.13.0-24.

Table 1 Hardware & Software Specification of Physical Servers

Processor	2.1 GHz Opteron
Number of CPU cores	12
Memory	32 GB
Disk Space	500 GB
Operating System	Ubuntu 14.04 64-bit

Note that the SM, CM, RA and PM components of the SIMaaS middleware reside in individual traditional virtual machines, each having 4 virtual CPUs, 8 GB memory and Ubuntu 14.04 64-bit operating system in our private cloud managed by OpenNebula 4.6.2. The SM was hosted on Cherrypy 3.6.0 web server. The CM uses Shipyard version v1 for managing Docker hosts along with custom python code we developed. The PM relies on a customized Perfmon Server Agent 2.2.3.RC1 residing on each Docker host to gather performance data. The RA utilizes ZeroMQ version 4.0.4 for receiving simulation results from the Docker containers.

5.2 Evaluating SIMaaS for Meeting Deadlines and Resource Consumption

We evaluate the ability of the SIMaaS middleware to the meet user-specified deadline and its effectiveness in minimizing the resources consumed. The DT-SHS used for the experiments provides the approximate number of simulations needed for stochastic model checking as an input to attain the desired confidence level for the output [33]. These studies were conducted for different resource overbooking ratios, number of simulations executed, deadlines, and simulation duration. Overbooking refers to the number of times the capacity of a physical resource is exceeded. For example, each container is assigned a single CPU; thus for a 12-core system, an overbooking ratio of 2 translates to 24 containers running on the host. This strategy is cost effective when the guest does not consume all the assigned resources. The experiments were conducted over six Docker hosts. We run the scheduling policy defined in Algorithm 1 at an interval of 0.5 secs that dynamically allocates extra hosts if the deadline cannot be met with the assigned hosts.

Test 1 – Varying Host Overbooking Ratios: This set of experiments were performed to measure the capabilities of the system to handle multiple parallel requests made to hosts with varying overbooking ratios, and study their performance while running the containers. Table 2 shows the results of the experiments conducted for performing statistical model checking with 1,000 simulations and a user-specified deadline of 300 seconds. Our DTSHS uses sampling rate as an input to the simulations that determines the amount of time simulation takes for execution. A 10 ms sampling rate in ideal conditions takes approximately 5 seconds to execute per simulation run for our use case.

Table 2 System Performance with Varying Host Overbooking Ratios

Overbooking Ratio	Hosts Acquired	Response Time to User (in secs)	Turnaround Time per Sim (in ms)	Measured Overhead(%)
0.5	5	251.7	6609.85	37.13
1	3	259.9	7525.41	56.63
2	2	254.3	9167.99	66.14
3	3	284.6	14604.72	84.73
4	2	272.1	16811.32	87.64
5	2	267.2	18396.4	91.15
6	2	269.2	19162.75	109.84
7	6	Exceeded Deadline	20838.18	121.38
8	6	Exceeded Deadline	16906.13	144.76

We measure the total number of hosts acquired by the system to meet the deadline. The system's goal is to minimize this number to keep the economic cost within the bounds. We also measure the response time observed by the system user after the system finds the desired solution, the average turnaround time per simulation from the instant it gets requested till the results get logged, and the system overhead with respect to the actual simulation execution time. This overhead includes the performance interference overhead, resource contention, time consumed in data transfer at different components of the workflow and also the time consumed in aggregation logic.

From the results we can conclude that for CPU-intensive applications – simulations tend to fall in this category – the non-overbooked system provides the best results, however, the number of hosts needed is also high, which in turn increases the economic cost. A highly overbooked system too has high cost and will be unable to meet the deadlines due to performance overhead and should be avoided. Based on empirical results, a lower overbooked scenario provides ideal trade-off as it needs less number of hosts and is able to meet the deadlines. We also note that the system overhead increases nearly linearly with the overbooking ratio as the contention increases.

Based on the experiments, we also illustrate the CPU utilization and memory utilization in Figures 3 and 4, respectively. The simulations have a low memory footprint but the CPU utilization is quite high. This conforms to our earlier result that having no or low overbooking for the host will provide better performance.

Test 2 - Varying Number of Simulations: The purpose of these tests is to demonstrate the scalability of SIMaaS middleware with increasing number

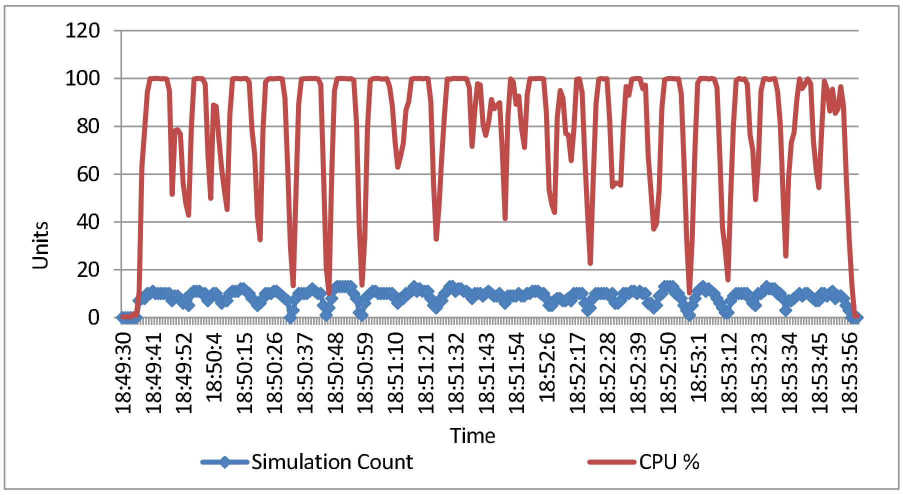

Fig. 3 CPU Utilization Variations with Simulation Count

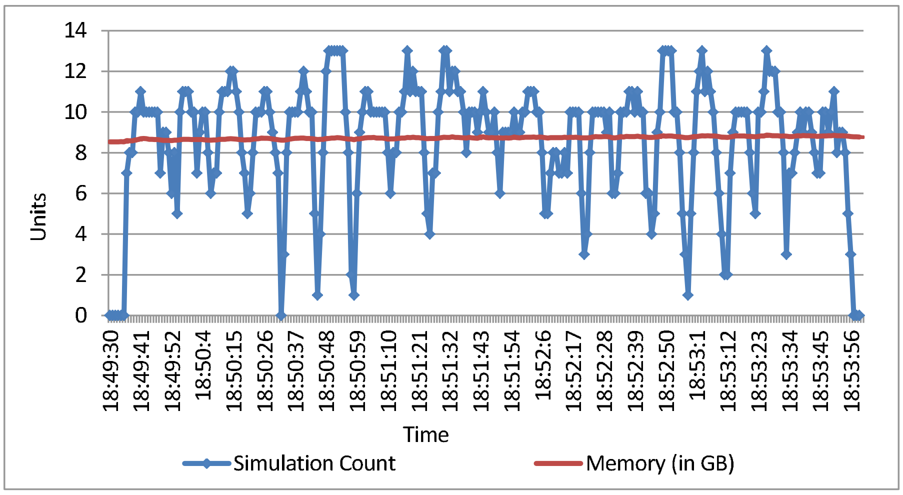

Fig. 4 Memory Utilization Variations with Simulation Count

of simulations that are needed as the fidelity of statistical model checking increases. The tests were run with a deadline of 600 seconds with a sampling rate of 10 ms and overbooking ratio of 2. Table 3 shows the results. The system is able to scale to 5,000 simulations without any additional overhead.

Test 3 - Varying Simulation Duration: For these experiments, we vary the sampling rate of our simulation model. Increasing the sampling period provides more accuracy for the results but also increases the simulation duration. Table 4 measures and presents the simulation performance for varying durations. We observe that the overhead reduces significantly as the duration of the

Table 3 System Performance with Varying Number of Simulations

Number of Simulations	Hosts Acquired	Response Time to User (in secs)	Turnaround Time per Sim (in ms)	Measured Overhead(%)
500	1	266.3	10317.5	91.75
1000	1	538.6	9873.18	93.12
2500	3	254.3	9607.45	79.03
5000	5	593.9	9625.59	78.88

container execution increases, which is attributed mainly to less percentage of time spent in scheduling and start up of containers.

Table 4 System Performance with Varying Simulation Duration

Sampling Rate (in ms)	Hosts Acquired	Response Time to User (in secs)	Turnaround Time per Sim (in ms)	Measured Overhead(%)
10	1	266.3	10317.5	91.75
5	2	234.6	19600.57	11.67
1	5	481.9	93664.98	2.65

Test 4 - Varying Simulation Deadline: Table 5 displays how the system is able to meet varying deadlines by scaling to consume higher resources but without adding any significant additional overhead per simulation.

Table 5 System Performance with Varying Simulation Deadline

Specified Deadline (in secs)	Hosts Acquired	Response Time to User (in secs)	Turnaround Time per Sim (in ms)	Measured Overhead(%)
600	1	538.6	9873.18	93.12
300	3	188.8	10071.29	74.34
150	5	122.4	10388.9	67.18
120	6	117.5	11112.11	92.14

5.3 Comparison of Linux Container-based Cloud and Traditional Cloud

This set of experiments affirm the large difference in startup time of container in Linux container-based cloud and virtual machine in hypervisor-based traditional cloud. In [21], the authors showed that there is a high start up time requirement on different popular public clouds. We tested the same on our private cloud, managed by OpenNebula and running QEMU-KVM hypervisor. We used overbooking ratios of 1, 2 and 4 with a minimal image. While the startup time were in the order of sub-seconds for our Linux container host, they were 176, 300 and 599 seconds, respectively on the hypervisor host. The large start up time can be ascribed to the time taken in copying the image to the VM and booting of the operating system.

Another set of experiments were performed to compare the performance of a host running simulations using Linux container versus virtual machines. Table 6 shows that Linux container host performs better in most of the cases as it does not have the overhead of running another operating system that a VM has.

Table 6 Comparison of Simulation Execution Time

	Overbooking Ratio 1	Overbooking Ratio 2	Overbooking Ratio 4
Linux Container (10 ms sampling)	4.74s	7.19s	13.32s
Virtual Machine (10 ms sampling)	5.17s	9.71s	19.05s
Linux Container (1 ms sampling)	50.5s	98.29s	180.45s
Virtual Machine (1 ms sampling)	52.4s	97.56s	202.5s

6 Conclusions

This paper described the design and empirical validation of a cloud middleware solution to support the notion of simulation-as-a-service. Our solution is applicable to those systems whose models are stochastic and require a potentially large number of simulation runs to arrive at outcomes that are within statistically relevant confidence intervals.

Many insights were gained during this research as follows and resolving these form the dimensions of our future investigations:

– Several competing alternatives are available to realize different aspects of cloud hosting. Effective use of software engineering design patterns is necessary to realize the architecture for cloud-based middleware solutions so that individual technologies can be swapped with alternate choices.
– Our empirical results suggested that an overbooking ratio of two and host counts of two provided the best configuration to execute the simulations. However, these conclusions were based on the existing use case and the small size of our private data center. Moreover, no background traffic was considered. Our future work will explore this dimension of the work as well as determine a mathematical bound for the optimal configuration.
– In our approach the number of simulations to execute for stochastic model checking were based on published results for the use case. In future there will be a need to determine these quantities through modeling and empirical means.

Acknowledgments.

This work was supported in part by the National Science Foundation CAREER CNS 0845789 and AFOSR DDDAS FA9550-13-1-0227. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF and AFOSR.

References

1. Abate A, Prandini M, Lygeros J, Sastry S (2008) Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. Automatica 44(11):2724–2734
2. Abate A, Katoen JP, Lygeros J, Prandini M (2010) Approximate model checking of stochastic hybrid systems. European Journal of Control 16(6):624–641
3. Al-Zoubi K, Wainer G (2011) Distributed simulation using restful interoperability simulation environment (rise) middleware. In: Intelligence-Based Systems Engineering, Springer, pp 129–157
4. Alamri A, Ansari WS, Hassan MM, Hossain MS, Alelaiwi A, Hossain MA (2013) A survey on sensor-cloud: architecture, applications, and approaches. International Journal of Distributed Sensor Networks 2013
5. Alur R, Pappas G (2004) Hybrid Systems: Computation and Control: 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings, vol 7. Springer
6. Atzori L, Iera A, Morabito G (2010) The internet of things: A survey. Computer networks 54(15):2787–2805
7. Bardac M, Deaconescu R, Florea AM (2010) Scaling peer-to-peer testing using linux containers. In: Roedunet International Conference (RoEduNet), 2010 9th, IEEE, pp 287–292, URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5541555
8. Calheiros RN, Vecchiola C, Karunamoorthy D, Buyya R (2012) The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds. Future Generation Computer Systems 28(6):861–870
9. Choi C, Seo KM, Kim TG (2014) Dexsim: an experimental environment for distributed execution of replicated simulators using a concept of single-simulation multiple scenarios. Simulation p 0037549713520251
10. Fujimoto RM (1990) Parallel discrete event simulation. Communications of the ACM 33(10):30–53
11. Fujimoto RM, Malik AW, Park A (2010) Parallel and distributed simulation in the cloud. SCS M&S Magazine 3:1–10
12. Gao Y, Wang Y, Gupta SK, Pedram M (2013) An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems. In: Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, IEEE Press, p 31

13. García-Valls M, Cucinotta T, Lu C (2014) Challenges in real-time virtualization and predictable cloud computing. Journal of Systems Architecture
14. Handigol N, Heller B, Jeyakumar V, Lantz B, McKeown N (2012) Reproducible network experiments using container-based emulation. In: Proceedings of the 8th international conference on Emerging networking experiments and technologies, ACM, pp 253–264, URL http://dl.acm.org/citation.cfm?id=2413206
15. Hellegouarch S (2007) CherryPy Essentials: Rapid Python Web Application Development. Packt Publishing Ltd
16. Kim H, El-Khamra Y, Rodero I, Jha S, Parashar M (2011) Autonomic management of application workflows on hybrid computing infrastructure. Scientific Programming 19(2):75–89
17. Ledyayev R, Richter H (2014) High performance computing in a cloud using openstack. In: CLOUD COMPUTING 2014, The Fifth International Conference on Cloud Computing, GRIDs, and Virtualization, pp 108–113
18. Li Z, Li X, Duong T, Cai W, Turner SJ (2013) Accelerating optimistic hla-based simulations in virtual execution environments. In: Proceedings of the 2013 ACM SIGSIM conference on Principles of advanced discrete simulation, ACM, pp 211–220
19. Liu X, He Q, Qiu X, Chen B, Huang K (2012) Cloud-based computer simulation: Towards planting existing simulation software into the cloud. Simulation Modelling Practice and Theory 26:135–150
20. LXC (2014) Linux container. URL https://linuxcontainers.org/, last accessed: 10/11/2014
21. Mao M, Humphrey M (2012) A performance study on the vm startup time in the cloud. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, IEEE, pp 423–430
22. Mauch V, Kunze M, Hillenbrand M (2013) High performance cloud computing. Future Generation Computer Systems 29(6):1408–1416
23. Menage PB (2007) Adding generic process containers to the linux kernel. In: Proceedings of the Linux Symposium, Citeseer, vol 2, pp 45–57, URL https://www.kernel.org/doc/ols/2007/ols2007v2-pages-45-58.pdf
24. Merkel D (2014) Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux J 2014(239), URL http://dl.acm.org/citation.cfm?id=2600239.2600241
25. Rak M, Cuomo A, Villano U (2012) Mjades: Concurrent simulation in the cloud. In: Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on, IEEE, pp 853–860
26. Shipyard (2014) Shipyard project. URL http://shipyard-project.com/, last accessed: 10/11/2014
27. Soltesz S, Pötzl H, Fiuczynski ME, Bavier A, Peterson L (2007) Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In: ACM SIGOPS Operating Systems Review, ACM, vol 41, pp 275–287, URL http://dl.acm.org/citation.cfm?id=1273025

28. Somasundaram TS, Govindarajan K (2014) Cloudrb: A framework for scheduling and managing high-performance computing (hpc) applications in science cloud. Future Generation Computer Systems 34:47–65
29. Tao F, Zhang L, Venkatesh V, Luo Y, Cheng Y (2011) Cloud manufacturing: a computing and service-oriented manufacturing model. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture p 0954405411405575
30. Vanmechelen K, De Munck S, Broeckhove J (2012) Conservative distributed discrete event simulation on amazon ec2. In: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), IEEE Computer Society, pp 853–860
31. Xavier MG, Neves MV, Rossi FD, Ferreto TC, Lange T, De Rose CA (2013) Performance evaluation of container-based virtualization for high performance computing environments. In: Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on, IEEE, pp 233–240, URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6498558
32. Zhu X, Chen H, Yang LT, Yin S (2013) Energy-aware rolling-horizon scheduling for real-time tasks in virtualized cloud data centers. In: High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on, IEEE, pp 1119–1126
33. Zuliani P, Platzer A, Clarke EM (2013) Bayesian statistical model checking with application to stateflow/simulink verification. Formal Methods in System Design 43(2):338–367