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

A Simulation as a Service Cloud Middleware

Shashank Shekhar, Hamzah Abdel-Aziz, Michael Walker, Faruk Caglar, Aniruddha Gokhale, Xenofon Koutsoukos

Received: date / Accepted: date

Abstract Many seemingly simple questions that individual users face in their daily lives may actually require substantial number of computing resources to identify the right answers. For example, a user may want to determine the right thermostat settings for different rooms of a house based on a tolerance range such that the energy consumption and costs can be maximally reduced while still offering comfortable temperatures in the house. Such answers can be determined through simulations. However, some simulation models as in this example are stochastic, which require the execution of a large number of simulation tasks and aggregation of results to ascertain if the outcomes lie within specified confidence intervals. Some other simulation models, such as the study of traffic conditions using simulations may need multiple instances to be executed for a number of different parameters. Cloud computing has opened up new avenues for individuals and organizations with limited resources to obtain answers to problems that hitherto required expensive and computationally-intensive resources. This paper presents SIMaaS, which is a cloud-based Simulation-as-a-Service to address these challenges. 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. Empirical results validating our claims are presented in the context of two case studies.

The final publication is available at Springer via `http://dx.doi.org/10.1007/s12243-015-0475-6`

Shashank Shekhar, Hamzah Abdel-Aziz, 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

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

1 Introduction

With the advent of the Internet of Things (IoT) paradigm [7], which involves the ubiquitous presence of sensors, there is no dearth of collected 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. Any service meant to find answers to these questions will very likely require substantial number of computing resources. Moreover, users will 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 quite complex to develop. Some models may be stochastic in nature, which require a large number of compute-intensive executions of the models to obtain outcomes that are within a desired statistical confidence interval. Other kinds of simulation models require running a large number of simulation instances with different parameters. Irrespective of the simulation model, individual users and even small businesses cannot be expected to acquire the needed resources in-house. Cloud computing then becomes an attractive option to host such services particularly because hosting high performance and real-time applications in the cloud is gaining traction [4, 32]. 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 [19].

Given these trends, it is important to understand the challenges in hosting such simulations in the cloud. To that end we surveyed prior efforts [17, 27, 28, 29] that focused on deploying parallel discrete event simulations (PDES) [16] in the cloud, which reveal that the performance of the simulation deteriorates as the size of the cluster distributed across the cloud increases. This occurs due primarily to the limited bandwidth and overhead of the time synchronization protocols needed in the cloud [42]. 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 still benefit from cloud computing. For example, complex system simulations that require statistical validation or those that compare simulation results under different constraints and parameter values often need to run repeatedly are suited to cloud hosting. 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 a large number of 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. Cloud computing becomes an attractive platform to host such capabilities [41]. To that end we have architected a cloud-based solution comprising resource management algorithms and middleware called Simulation-as-a-Service (SIMaaS).

It is possible to realize SIMaaS on top of traditional cloud infrastructure, which 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 a large number of multiple, short-duration simulations in parallel, the considerable setup and tear down overhead imposed by VMs, as demonstrated in Section 5.2, is unacceptable. Likewise, a solution based on maintaining a VM pool that is used by many cloud resource management frameworks such as [24, 12, 44, 18] is not suitable either since it can lead to resource wastage and may not be able to cater to sudden increases in service demand. Thus, a lightweight solution is desired.

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

- We propose a cloud middleware for SIMaaS that leverages Linux container [30]-based infrastructure, which has low runtime overhead, higher level of resource sharing, and very low setup and tear down costs.
- We present a resource management algorithm, that reduces the cost to the service provider and enhances the parallelization of the simulation jobs by fanning out more instances until the deadline is met while simultaneously auto-tuning itself based on the feedback.
- We show how the middleware intelligently generates different configurations for experimentation, and intelligently schedules the simulations on the Linux container-based cloud to minimize cost while enforcing the deadlines.
- Using two case studies, we show the viability of a Linux container-based SIMaaS solution, and illustrate the performance gains of a Linux container-based approach over 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 them with our contributions; Section 3 provides two use cases that drive the key requirements that are met by our solution; 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 dimensions: simulations hosted

in the cloud, cloud frameworks that provide resource management with deadlines, and container-based approaches. These dimensions of related work are important because 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 [35] 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 [15] 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 remote management 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 Linux containers that are more efficient and more suitable to the kinds of simulations hosted by SIMaaS than the VM-based approaches used by these solutions.

CloudSim [11] is a toolkit for modeling and simulating VMs, data centers and resource allocation policies without incurring any cost, which in turn helps to measure the feasibility and tune the performance bottlenecks. EMUSIM [13] enhances CloudSim by integrating an emulator to achieve the same purpose. SimGrid [14] is another distributed systems simulator used to improve the algorithms for data management infrastructure. We believe that the contributions of SIMaaS are orthogonal to these work. These related projects provide the platforms to evaluate resource allocation algorithms in the cloud while SIMaaS is a concrete realization of infrastructure middleware that supports different resource allocations. SIMaaS can benefit from these related work where resource management algorithms can first be evaluated in these platforms, and then deployed in the SIMaaS middleware. Additionally, we believe these related work do not yet support support Linux container based simulation of the cloud.

2.2 Related Work on Cloud Resource Management

There has been some work in cloud resource management to meet deadlines. Aneka [12] is a cloud platform that supports quality of service (QoS)-aware provisioning and execution of applications in the cloud. It supports different programming models, such as bag of tasks, distributed threads, MapReduce, actors and workflows. Our work on SIMaaS applies an advanced version of a resource management algorithm that is used by Aneka in the context of our Linux container-based lightweight virtualization solution. Aneka 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.

Another work close to our resource allocation policy is [10] that employs a cost-efficient scheduling heuristics to meet the deadline. However, this work is sensitive to execution time estimation error, whereas our work self-tunes based on feedback.

CometCloud [24] 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 [40] 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. [44] employed a rolling-horizon optimization policy to develop an energy-aware cloud data center for real-time task scheduling. All these efforts provide scheduling algorithms to meet deadlines on virtual machine-based cloud platforms 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. We also apply a heuristic based feedback mechanism to ensure deadlines are met with minimum resources.

In prior work [37, 26], we have designed and deployed multi-layered resource management algorithms integrated with higher-level task (re-)planning mechanisms to provide performance assurances to distributed real-time and embedded applications. These algorithms were integrated within middleware solutions that were deployed on a distributed cluster of machines, which can be viewed as small-scale data centers. These prior works focused primarily on affecting the application, such as migrating application components, load balancing, fault tolerance, deployment planning and to some extent scheduling. We view these prior works of ours as complementary to the current work. In the present work, we are more concerned with allocating resources on-demand. A more significant point of distinction is that the prior works focused on distributed applications that are long running while in current work we are focusing on applications that have a short running time but where we need to execute a large number of copies of the same application.

2.3 Related Work using Linux Containers

The Docker [34] 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 [33] and benchmark testing [39] 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) [43], reproducible network experiments [21], and peer-to-peer testing environments [8]. The demands and goals of each of these three efforts focus 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 Cases and Key Requirements for SIMaaS

We now present two use cases belonging to systems modeling that we have used in this paper to bring out the challenges that SIMaaS should address, and to evaluate its capabilities.

3.1 System Modeling Use Cases

System modeling for simulations is a 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. We use two use cases to highlight the different types of simulations that SIMaaS is geared to support.

3.1.1 Use Case 1: The Multi-room Heating System

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

We discuss here a DTSHS model of a multi-room heating system [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 has at most one heater at a time. Moreover, each room has its own user setting (i.e., constraints) for temperature. However, the rooms have an exchangeable effect with their adjacent rooms and with the ambient temperature.

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 the individual heater, which can be in ON or OFF state, and the continuous state is the room temperature. A discrete transition function switches the heater's status in each room based on using a typical controller which switches the heater on if the room temperature gets below a certain threshold xl and switches the heater off if the room temperature exceeds a certain threshold xu.

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. Typical system requirements that can be evaluated using simulations 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).

In our model of the system, we have used one of many possible strategies 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 is control threshold used to determine when room i needs to acquire a heater. The simulation model for this use case uses statistical model checking by Bayesian Interval Estimates [45].

3.1.2 Use Case 2: Traffic Simulation for Varying Traffic Density

Use case 2 targets transportation researchers and traffic application providers, such as Transit Now (`http://transitnownashville.org/`), who want to model and simulate different traffic scenarios within a relevant time window but do not have sufficient resources to do it in-house. We motivate this use case with a microscopic traffic simulator called SUMO [9] that can simulate city level traffic. The simulator can import a city map in OpenStreetMap [20] format to its own custom format. The user can supply various input parameters such as number of vehicles, traffic signal logic, turning probability, maximum lane speed and study their impact on traffic congestion.

One such "what if" scenario involves the user changing the number of vehicles moving in a particular area of the city and studying its impact. In contrast to use case 1 where all the stochastic simulation instances had nearly the same execution time in ideal conditions, in this use case, the simulation execution time varies with the input number of vehicles. Figure 1 illustrates how the execution time varies with the number of vehicles for a duration of 1000 seconds.

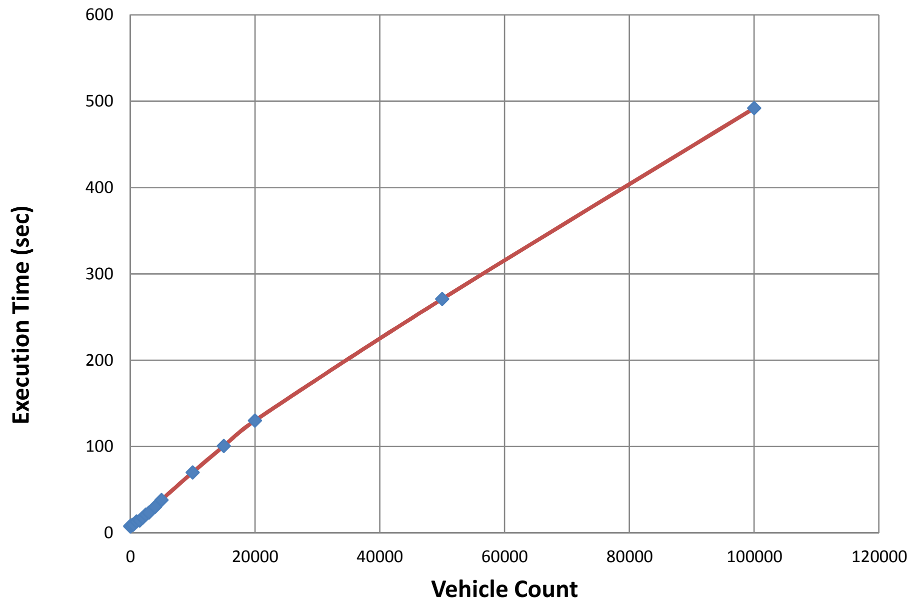


Fig. 1: Simulation Execution Time

3.2 Problem Statement and Key Requirements for SIMaaS

Based on the two use cases described above, we now bring out the key requirements that must be satisfied by SIMaaS. Addressing these requirements forms the problem statement for our research presented in this paper.

• **Requirement 1: Ability to Elastically Execute Multiple Simulations** – Recall that the simulation model for use case 1 is stochastic, which means that every simulation execution instance may yield a different simulation trajectory and results. To overcome this problem, we have to use the *statistical model checking (SMC)* approach based on Bayesian statistics [44, 45]. 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 several different simulation trajectories of the model and dividing the number of satisfied trajectories (i.e., true properties) over the total number of simulations. Thus, SMC requires execution of a large number of simulation tasks.

On the other hand, although the simulation models for use case 2 are not stochastic, the result of the simulation will often be quite different depending on the parameters supplied to the model. For example, varying the number of vehicles on the road, number of traffic lights, number of lanes, and speed limits will all generate different results. A user may be interested in knowing the results for various scenarios, which in turn requires a number of simulations to be executed seeded with different parameter values. In addition, the service will

be used by multiple users who need to execute different number of simulations, which is not known to the system a priori. This requirement suggests the need to elastically scale the number of simulation instances to be executed.

In summary, the two use cases require that SIMaaS be able to elastically scale the number of simulations that must be executed.

• **Requirement 2: Bounded Response Time** – In both our use cases, the user expects that the system respond to their requests within a reasonable amount of time. Thus, the execution of a large number of simulations that are elastically scheduled on the cloud platform, and result aggregation must be accomplished within a bounded amount of time so that it is of any utility to the user. Moreover, use case 2 illustrates an additional challenge that requires estimating the expected execution time for previously unknown parameters and ensuring that the system can still respond to user request in a timely manner.

In summary, SIMaaS must ensure bounded response times to user requests.

• **Requirement 3: Result Aggregation** – Both our use cases highlight the need for result aggregation. In use case 1, there is a need to aggregate the results from the large number of model executions to illustrate the confidence intervals for the results. In use case 2, the user will need a way to aggregate results of each run corresponding to the parameter values. Since SIMaaS is meant to be a broadly applicable service, it will require the user to supply the appropriate aggregation logic corresponding to their needs.

In summary, SIMaaS needs an ability to accept user-supplied result aggregation logic, apply it to the results of the simulations, and present the results to the user.

• **Requirement 4: Web-based Interface** – Since SIMaaS is envisioned as a broadly applicable cloud-based, simulation-as-a-service, it will not know the details of the user's simulation model. Instead, it will require the user to supply a simulation model of their system and various parameters to indicate how SIMaaS should run their models. For example, since use case 1 requires stochastic model checking, it will require a large number of simulation trajectories to be executed. Thus, SIMaaS will require the user to supply the simulation image and specify how many such simulations should be executed, the building layout, the number of heaters, the strategy used and so on. Similarly, for use case 2, SIMaaS will need to know how the model should be seeded with different parameter values and how they should be varied, which in turn will dictate the number of simulations to execute and their execution time. Finally, the aggregated results must somehow be displayed to the user.

In summary, SIMaaS should provide a web-based user interface to the users so they can supply both the simulation model and the parameters as well as receive the results using the interface.

4 SIMaaS Cloud Middleware Architecture

A cloud platform is an attractive choice to address the requirements highlighted in Section 3.2 because it can elastically and on-demand execute the multiple different simulation trajectories of the simulation models in parallel, and perform aggregation such as 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. To that end we have architected the SIMaaS cloud-based simulation-as-a-service and its associated middleware as shown in Figure 2. The remainder of this section describes the architecture and shows how it addresses all the requirements outlined earlier.

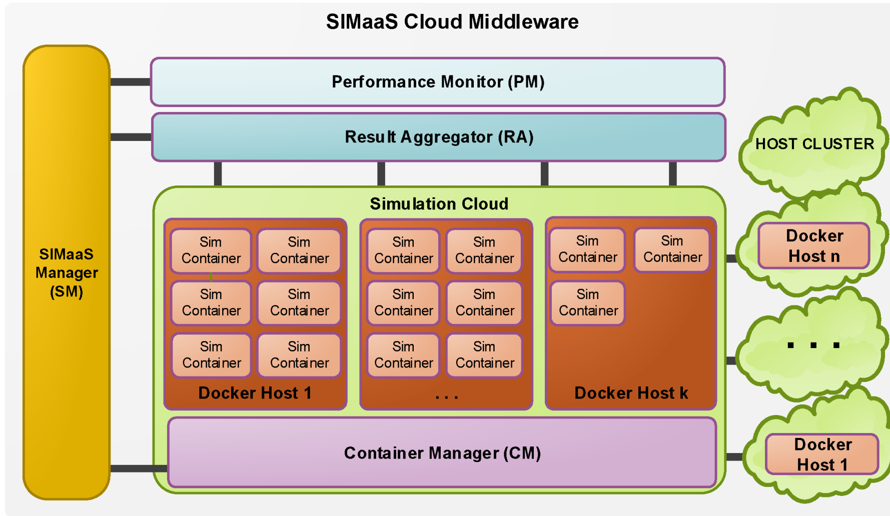


Fig. 2: System Architecture

4.1 Dynamic Resource Provisioning Algorithm: Addressing Requirements 1 and 2

Requirement 1 calls for elastic deployment of a large number of simulation executions depending on the use case category, which needs dynamic resource management. Requirement 2 calls for timely response to user requests. Thus, the dynamic resource management algorithm should be geared towards meeting the user needs.

For this paper, we define a QoS-based resource allocation policy that allocates containers for each requested simulation model such that its deadline is met and its cost, i.e. the number of assigned containers is minimized. We

assume that the user provides the following inputs to our allocation algorithm: simulation model, number of simulations and their corresponding simulation parameters, and the estimated execution time using some of the simulation parameters.

Formally, we define the requested execution of the simulation model as a *job*. Each job is made up of several different tasks representing an execution instance of that simulation job. At any instant in time k, $J(k)$ is the set of jobs which our allocation algorithm handles. Furthermore, for the jth job, $J_j(k) \in J(k)$, we define its deadline as DL_j, the number of containers it uses as $B_j(k)$, its ith simulation task as $T_{ij}(k)$, the simulation parameter of its ith task as θ_{ij} and finally, the expected execution time of its ith task with its corresponding parameter θ_{ij} as $E_{\theta_{ij}}(T_{ij}(k))$.

The primary objective of our allocation algorithm is to minimize the resource usage cost considered in terms of the number of containers used to serve the user simulation request, while maintaining the user constraint stated as meeting the deadline. To formalize this objective, we define it as the following optimization problem:

$$\begin{aligned} \min_{B_j} \quad & c(K) = \sum_j \sum_k B_j(k) = c(K-1) + \sum_j B_j(K) \\ \text{subject to} \quad & \forall j \in J(k), \frac{R_j(k)}{B_j(k)} = \frac{\sum_i E_{\theta_{ij}}(T_{ij}(k))}{B_j(k)} \leq DL_j \end{aligned}$$

where, $c(k)$ is the cost function at time instant k, and $R_j(k)$ is the jth job's total execution time which is equal to the summation of the execution time $E_{\theta_{ij}}(T_{ij}(k))$ for all the unserved tasks $T_{ij}(k)$. This constraint equation calculates the total time a job would take to finish if its simulations' executions have been parallelized using $B_j(k)$ containers. Therefore, it bounds the selection of $B_j(k)$ such that each job finishes before or by the deadline. To tackle this problem, we developed a simple heuristic shown in Algorithm 1 for efficiently selecting the minimum $B_j(k)$ such that each job finishes its required simulation tasks by their deadline.

Formally, we calculate B_j using the following formula. To simplify the notation, we will omit the time index k throughout the remainder of this section:

$$B_j = \frac{R_j}{DL_j}$$

Two major challenges arise when calculating B_j based on the above formula. First, it is difficult to calculate analytically R_j in a mathematically closed form because a task's execution time varies based on many dynamic factors such as performance interference, overbooking ratio, etc. Second, the execution times of a job's tasks are not necessarily identical when they have different simulation parameter θ_{ij}, as demonstrated in Figure 1. Therefore, the above formula is not accurate and we may need to increase the value of B_j calculated above in order to meet the deadline. Moreover, scheduling the job's tasks T_{ij} in the reserved containers B_j is a non-trivial task.

```
Input: J, α
while TRUE do
    Wait for(max(feedback_event, minimum_period));
    foreach  J_j ∈ J do
        // Update only jobs with new feedback data
        if  (HasFeedback(J_j)) then
            // Update the estimated error factor
            F_j ⟵ UpdateErrorFactor(E*_θij(T_ij)) ;
            // Update the estimated execution time function
            UpdateExecutionTimeFunction(F_j);
            // Update the number of containers with their scheduling
            extraContainersNeeded ⟵BestFitDecreasing(DL_j) - B_j;
            if extraContainersNeeded > 0 then
                Reserve(extraContainerNeeded, J_j);
            // Avoid frequent resource allocation and de-allocation
            else if extraContainersNeeded < threshold then
                Release(extraContainerNeeded, J_j);
            end
        end
    end
end
```

Algorithm 1: QoS-based Resource Allocation Policy

To overcome the first challenge, we make our heuristic calculation of B_j based on an estimated execution time $E'_{\theta_{ij}}(T_{ij})$ of each task T_{ij} and periodically update this estimation and consequently the B_j calculation based on the feedback of the actual executed time $E^*_{\theta_{ij}}(T_{ij})$. This simple feedback mechanism allows us to maintain our algorithm objective mentioned above while tolerating the effect of estimation errors, and handling the dynamic change of our system environment (e.g. performance interference). Furthermore, our algorithm has to wait for at least a minimum period of time even if there is a new feedback, in order to enhance the algorithm's performance by avoiding high frequent recalculation. In addition to this, we recalculate F_j and $E'_{\theta_{ij}}(T_{ij})$ in every iteration and the algorithm is executed only for jobs that have new feedback data.

In order to estimate the execution time R_j of the jth job, we use an initial execution time function $E_{\theta_{ij}}(T_{ij})$ as an initial function to our estimator. Since the user provides the estimated execution time for only a few parameters, we use a regression algorithm based on sequential exponential kernel [36] to build the initial function of $E_{\theta_{ij}}(T_{ij})$ using the data point provided by the user. Then, we update this function by an error factor F_j calculated using the feedback data. The calculation of the error factor F_j and the estimated execution time function $E'_{\theta_{ij}}(T_{ij})$ are shown in the following equation:

$$E'_{\theta_{ij}}(T_{ij}) = E_{\theta_{ij}}(T_{ij}) \times (1 + F_j)$$

such that:

$$F_j = \mathbb{E}[\Delta E_{\theta_{ij}}(T_{ij})/E_{\theta_{ij}}(T_{ij})] + \alpha \times \sqrt{Var\left[\Delta E_{\theta_{ij}}(T_{ij})/E_{\theta_{ij}}(T_{ij})\right]}$$

$$\Delta E_{\theta_{ij}}(T_{ij}) = E_{\theta_{ij}}(T_{ij}) - E^*_{\theta_{ij}}(T_{ij})$$

where $\alpha \geq 0$ is an estimator parameter that determines how pessimistic is our estimator because F_j covers more errors as α increases. For example, when $\alpha = 1, 2, 3, F_j$ will approximately estimate the worst-case scenario of 68%, 95%, and 99.7% of the feedback error values, respectively. For a real-time implementation of error factor calculation, we use an online algorithm developed by Knuth [25] to calculate $\mathbb{E}[.]$ and $Var[.]$ incrementally, in order to avoid saving and inspecting the entire feedback data every time a new feedback entry has arrived.

To overcome the second challenge, we used best fit decreasing bin packing algorithm [23], where, we pass DL_j to this bin packing algorithm as its bin's size input, and the estimated task execution times $E'_{\theta_{ij}}(T_{ij})$ as its items' size input. Therefore, the number of slots produced by the bin packing algorithm represents the required containers B_j and the distribution of the tasks T_{ij} in each slot represents the tasks' schedule over the containers.

Note that the system resources constraints limit the number of jobs in J that can be serviced at the same time. Therefore, we use an admission control algorithm shown in Algorithm 2 to maintain a reliable service. The admission control algorithm is used to accept any new incoming user request which can be handled using the remaining available resources without missing its and other running jobs deadline. It basically estimates the number of containers needed to serve the new request such that it finishes by its deadline. Then, it checks whether there are idle containers available to serve it or not. The algorithm has two administrator configurable parameters ($\beta \geq 1$) and ($\gamma \geq 0$) which add a margin of resources to overcome the estimation error and to maintain another margin of resources for other running jobs to be used by the above allocation algorithm.

Input: availableCapacity, newJob
Output: Accepted/Rejected
containersNeeded ⟵BestFitDecreasing(DL_{newJob});
if *containersNeeded* $\times\beta <$ *availableCapacity* $-\gamma$ **then**
  Accept newJob;
else
  Reject newJob;
end

Algorithm 2: Admission Control Algorithm

4.2 Dynamic Resource Provisioning Middleware: Addressing Requirements 1 and 2

The second aspect of dynamic resource management is the middleware infrastructure that encodes the algorithm and provides the service capabilities. The middleware aspect is described here.

4.2.1 Architectural Elements of the SIMaaS Middleware

The central component of the SIMaaS middleware shown in Figure 2 that is responsible for resource provisioning and handling user requests is the SIMaaS Manager (SM). All the coordination and decision making responsibilities are controlled by this component. It employs the strategy design pattern; thus it has a pluggable design that is used to strategize the virtualization approach to be used by the hosted system. The strategy pattern also allows the SM to swap the scheduling policy if needed, however, we use a single scheduling policy during the life-cycle of SIMaaS to avoid conflicts.

A cloud platform typically uses virtualized resources to host user applications. Different types 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 scheduled, this boot up time incurs a delay in availability of new VMs, not to mention the cost of the application's initialization time. All of these impact the user response time. Since Requirement 2 calls for bounded response time, SIMaaS uses the lightweight containers, which suffice for our purpose.

The life cycle of these containers is managed by the Container Manager (CM) shown in the Figure 2. The pluggable architecture of SM allows CM to switch between various container providers, which can be Linux container or hypervisor-based VM cloud. The Linux container is the default container provider of CM. Specifically, we use the Docker [34] 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 tearing 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 [38] for communicating with the Docker hosts, however, due to sluggish performance we observed, we had to implement a custom solution with a reduced role for Shipyard. Overcoming the reasons for the sluggish performance and reusing existing artifacts maximally is part of our future investigations when we also evaluate other container managers such as Apache Mesos, Google Kubernetes and Docker Swarm.

4.2.2 Resource Instrumentation and Monitoring

Recall that meeting user-specified deadlines is an important goal for SIMaaS (Requirement 2). These deadlines must be met in the context of either the stochastic model checking that requires multiple simultaneous runs of the stochastic simulation models or simulations executed under a range of parameter values. Thus, SIMaaS must be cognizant of overall system performance

so that our resource allocation algorithm can make effective dynamic resource management decisions. To support these system requirements, effective system instrumentation is necessary.

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 a 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 it for gathering system metrics. The metrics available are classified into eight 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 metrics, but they are available for use if needed. Each of these catagories have parameters to allow customization of the 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's performance precisely.

4.3 Result Aggregation: Addressing Requirement 3

Stochastic model checking as in use case 1 requires that results of the multiple simulation runs be aggregated to ascertain if the specified probabilistic property is met or not. Similarly, as in use case 2, multiple simulation runs for different simulation parameters result in different outcomes, which must be aggregated and presented to the user. To accomplish this and thereby satisfy Requirement 3, a key component of our middleware is the Result Aggregator (RA). RA receives the simulation results from the Docker containers. It uses ZeroMQ messaging queue service for reliable result delivery. It has two roles: first, it sends feedback to the SM about the completion of task for decision making. Second, it performs the actual result aggregation.

Since the aggregation logic is application-dependent, it is supplied by the user when the service is hosted, and is activated when the simulation job completes. For use case 1, the aggregation logic is a Bayesian statistical model checking which produces a single string result. On the other hand, use case 2 aggregation logic parses and collates the XML files produced as the result of simulation runs.

4.4 Web Interface to SIMaaS - SIMaaS Workflow and User Interaction: Addressing Requirement 4

Finally, we discuss how the user interacts with SIMaaS, which is a web-based interface, and the workflow triggered by a typical user action. The interface to the Simulation Manager of SIMaaS is hosted on a lightweight web server, CherryPy [22] to interact with the user and also to receive feedback from other SIMaaS components. The interaction involves two phases. In the design phase a user interacts with the SIMaaS interface and provides the initial configuration which includes the simulation executables and the aggregation logic. A container image is generated after including hooks to send the temporary results. This image is then uploaded to a private cloud registry accessible to the container hosts. The aggregation logic is deployed in the Result Aggregator component that can collect the temporary simulation results and generate the final response.

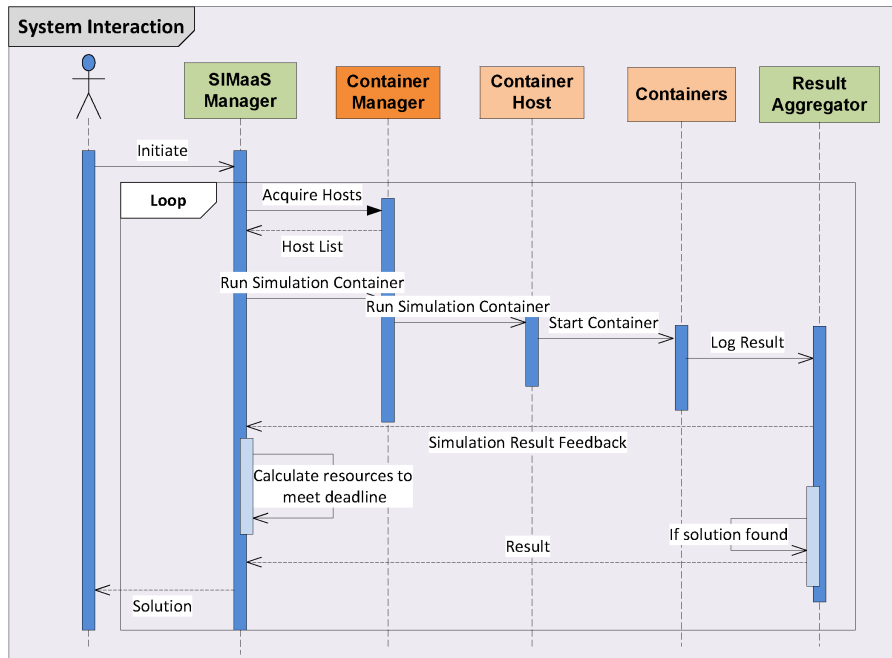


Fig. 3: SIMaaS Interaction Diagram

The execution phase is depicted in Figure 3, wherein, time bounded, on-demand simulation jobs are performed. The user can use a RESTful API (or a web-form if deadline is not immediate) to supply name-value pairs of parameters. The following parameters are supplied by all the users:

- Simulation Model Name: A simulation model name is required to identify the container image and aggregation logic.
- Number of Simulations: The number of simulation instances to run.
- Deadline: The deadline for the job
- Required Resources: Number of CPUs to be allocated for each container. Other types of resources will be added in future.
- Simulation Command: The command to initiate the simulation.
- List of (Execution Time Parameter, Estimated Execution Time): This is a small set of the execution time parameter values and the corresponding execution times that is used to generate the regression curve for estimating the unknown execution time for remaining parameters.

The following parameters are specific to the use case:

- Use Case 1 - Number of Heaters: The number of heaters active in the building (explained in section 3.1.1)
- Use Case 1 - Sampling Rate: The rate at which data is sampled for temperature simulation.
- Use Case 1 - Strategy: The model strategy to be used.
- Use Case 1 - Confidence Level: A confidence value for the Bayesian Aggregator (explained in section 3.1.1).
- Use Case 2 - SUMO Configuration File: A configuration file used by SUMO simulation for selecting the inputs and deciding the output details.
- Use Case 2 - List of Vehicle Counts: Different vehicle counts to be simulated.

We note that the simulation execution time is also a user input, however, this value can be determined in a sandbox environment, wherein executing a single simulation instance gives the value for a constant time simulation (such as use case 1) or by executing a subset of simulation instances from a different range of parameters (such as use case 2) and using a regression curve to estimate the execution time for others.

The request is then forwarded and processed by the SM. It validates the input and applies admission control as explained in Algorithm 2 using a resource allocation and scheduling policy, and checks if sufficient resources are available. If not, then it immediately responds to the user with a failure message. In future, based on the criticality of the request, some jobs may be swapped for a higher priority job. If the job can be scheduled, then it allocates the resources and contacts the CM to run 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. The SM also runs a service, applying Algorithm 1 at a configurable interval, to determine if the deadline will be met based on the current performance data, and accordingly contacts the CM to acquire additional resources and run the containers. Once the simulation completes, the RA responds to the user with the result. Currently it uses a shared folder. However, going forward we plan to implement either an interface that can send the response as an asynchronous callback or send a notification to the user about the availability of the result.

5 Experimental Validation

This section evaluates the performance properties of the SIMaaS middleware and validates our claims in the context of hosting the use cases described in Section 3.1.

5.1 Experimental Setup

Our setup consists of ten physical hosts each with the configuration defined in Table 1. The same set of machines were used for experimenting with both Linux containers and virtual machines. Docker version 1.6.0 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

Even though our solution is designed to leverage the Linux containers instead of virtual machines, since we did not have access to large number of physical machines yet had to measure the scalability of our approach, we tested our solution over a homogeneous cluster of 60 virtual machines deployed as docker hosts for running the simulation tasks, i.e., the docker containers were spawned inside the VMs. The same set of physical machines were used to host the VMs with configuration as defined in Table 2.

Table 2: Configuration of VM Cluster Nodes

Kernel	Linux 3.13.0-24
Hypervisor	Qemu-KVM
Number of Virtual Machines	6
Overbooking Ratio	2.0
Guest CPUs	4
Guest Memory	4 GB
Guest OS	Ubuntu 14.04 64-bit

Note that the SM, CM, RA and PM components of the SIMaaS middleware reside in individual virtual machines deployed on a separate set of hosts, each with 4 virtual CPUs, 8 GB memory and running Ubuntu 14.04 64-bit operating system in our private cloud managed by OpenNebula 4.6.2. The

Simulation Manager was deployed on CherryPy 3.6.0 web server. The container manager used Shipyard version v2 for managing the docker hosts. The performance monitor relied on a customized Perfmon Server Agent 2.2.3.RC1 residing in each docker host to collect performance data. The Result Aggregator utilized ZeroMQ version 4.0.4 for receiving simulation results from the docker containers.

5.2 Validating the Choice of Linux Container-based SIMaaS Solution

We first show why we used the container-based approach in the SIMaaS solution instead of traditional virtual machines. This set of experiments affirm the large difference in startup times for containers in Linux container-based cloud and virtual machines in hypervisor-based traditional cloud. In [31], the authors showed that there is a high start up time required on different popular public clouds. We tested similar configurations in our private cloud, managed by OpenNebula and running QEMU-KVM hypervisor. We used overbooking ratios of 1, 2 and 4 with a minimal image from use case 1. While the startup time were in the order of sub-seconds for our Linux container host, they were 176, 300 and 599 seconds, respectively, for the hypervisor host. The large start up time can be ascribed to the time taken in cloning the image as the VM file system and booting up 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 3 shows that the Linux container host performs better in most of the cases as it does not incur the overhead of running another operating system as a VM does.

Table 3: Comparison of Simulation Execution Time

	Overbooking Ratio 1	Overbooking Ratio 2	Overbooking Ratio 4
Linux Container (Physical Server - 4s)	4.74s	7.19s	13.32s
Virtual Machine (Physical Server - 4s)	5.17s	9.71s	19.05s
Linux Container (Physical Server - 50s)	50.5s	98.29s	180.45s
Virtual Machine (Physical Server - 50s)	52.4s	97.56s	202.5s

5.3 Workload for Container-based Experimentation

The workload we used in our experiments consists of several jobs corresponding to user requests, each having a number of simulation instances as a bag

of independent tasks. The simulations are containerized as docker images on Ubuntu 14.04 64-bit operating system. The jobs are based on both the use cases described in section 3.1, however, we created several variations of these use cases by changing the execution parameters. The building heating stochastic simulation jobs have near constant execution time, but we used three variations of it by using different sampling rates of 10, 5, and 2 milliseconds. The smaller the sampling rate, better is the accuracy of the simulation results at the expense of longer execution times. The traffic simulations jobs also had several variations based on the range of the vehicle count.

The simulations for each job may have different resource requirements that will be provided by the user. For these experiments, we have considered CPU-intensive workloads and modeled the user input as three resource types with 1, 2 or 4 CPUs per container, which is an indication of how much CPU share that container gets. Thus, we convert these values to CPU share per host as the docker container input.

We generated a synthetic workload to measure the system performance. We conducted two sets of experiments. The first set of experiments were conducted to measure the efficacy of our algorithm using a single type of job on a ten physical host cluster, whereas the second set of experiments were performed to demonstrate the scalability of our algorithm using a 60 virtual-host cluster with different types of jobs arriving at different points in time. We applied the Poisson distribution with λ of 1 for a duration of two hours to find the job arrival distribution. The number of tasks per job was uniformly distributed from 100 to 500. The deadline per job was also varied as a uniform distribution from 5 minutes to 20 minutes.

5.4 Evaluating SIMaaS for Meeting Deadlines and Resource Consumption

We evaluate the ability of the SIMaaS middleware to meet the user-specified deadline and its effectiveness in minimizing the resources consumed. In use case 1 described in section 3.1.1, the user provides the approximate number of simulations needed for stochastic model checking as an input to attain the desired confidence level for the output [45]. For the second use case describe in 3.1.2, the number of simulations is a user input. These studies were conducted for different resource overbooking ratios, simulation count, deadlines, simulation duration, and execution times. Overbooking refers to the number of times the capacity of a physical resource is exceeded. For example, suppose 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 guests do not consume all the assigned resources all at the same time. We run the scheduling policy defined in Algorithm 1 at an interval of 2 secs that dynamically allocates extra hosts if the deadline cannot be met with the assigned hosts.

Test 1 – Determining the Error Estimation Parameter (α)*:* These experiments were performed on the ten physical host cluster with use case 1 as the simulation model with the following parameters: a deadline of 2 minutes, 500 simulation tasks, one CPU per container and overbooking ratio of 2 per host. The purpose of these tests was to determine the error estimation parameter – α for our system. This value is used to calculate the error factor, explained in Section 4.1, that plays a crucial role in meeting the deadline and allocating container slots. Recall also that our algorithm attempts to minimize the number of containers while meeting deadlines on a per simulation job basis.

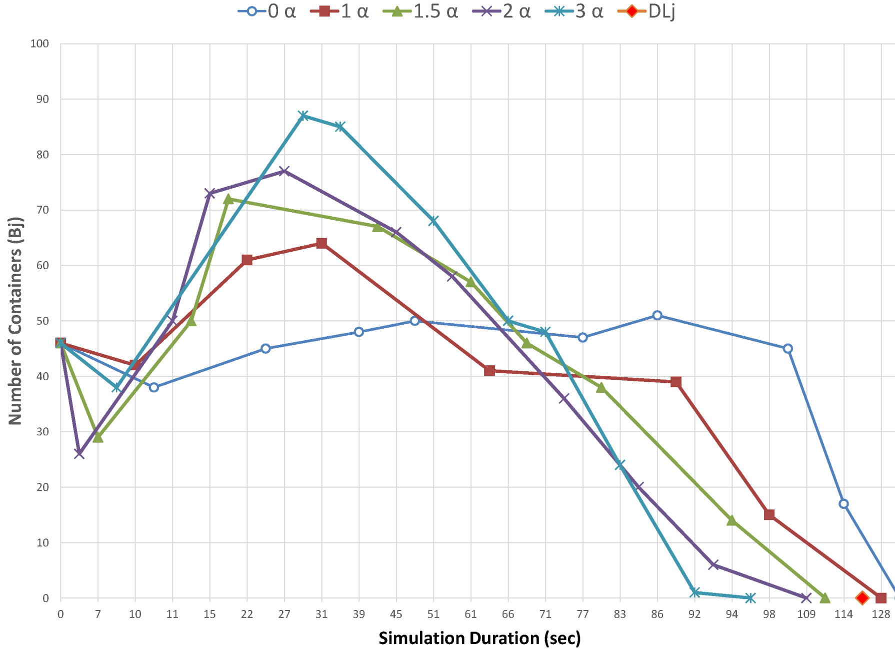
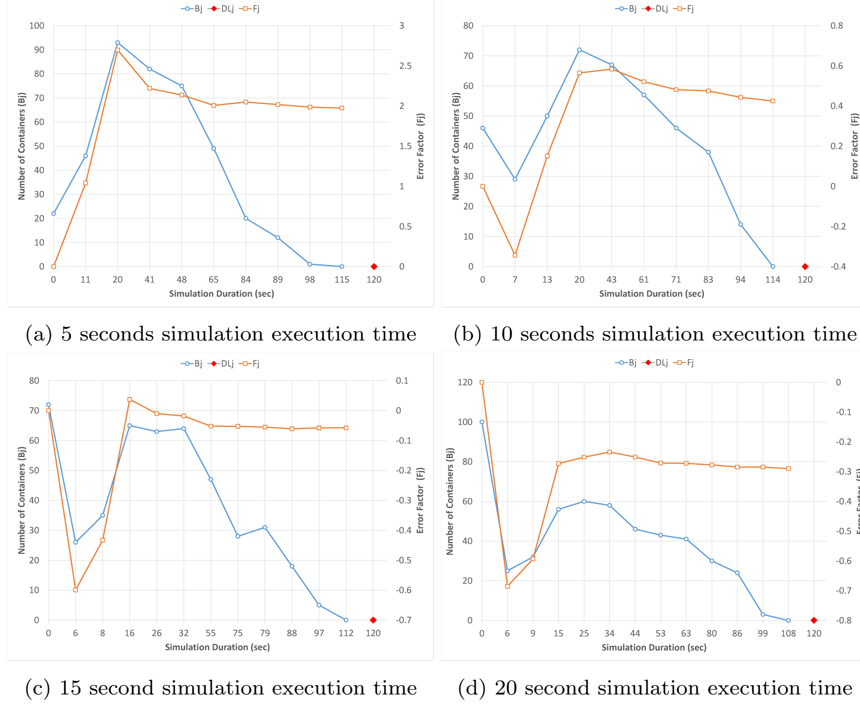


Fig. 4: Container Count and Deadline Variation with α

Figure 4 depicts the simulation results for α values 0, 1, 1.5, 2 and 3. We observe that values of 0 and 1 were too low and the system missed the deadline. A value of 1.5 was found to meet the deadline as well as causing less peak resource usage. This value can be made dynamic based on the user urgency and strictness of the deadline.

Test 2 – Studying Variations in Container Count (B_j) *with Error Factor* (F_j)*:* These tests were conducted to study the impact of changing the feedback based error factor (F_j) on the container count (B_j) as well as the input simulation execution time. From Figure 5, we observe that initially, the resource consumption varies according to the error factor. Later, the resource consumption stays constant after the error estimate reaches a steady state. We conclude that the

error estimate becomes accurate and close to the real value we get from the feedback. However, as the simulation moves towards completion, resources get released as lesser number of simulations remain to be executed.



(a) 5 seconds simulation execution time

(b) 10 seconds simulation execution time

(c) 15 second simulation execution time

(d) 20 second simulation execution time

Fig. 5: Variation in Error Factor (F_j) and Container Count (B_j)

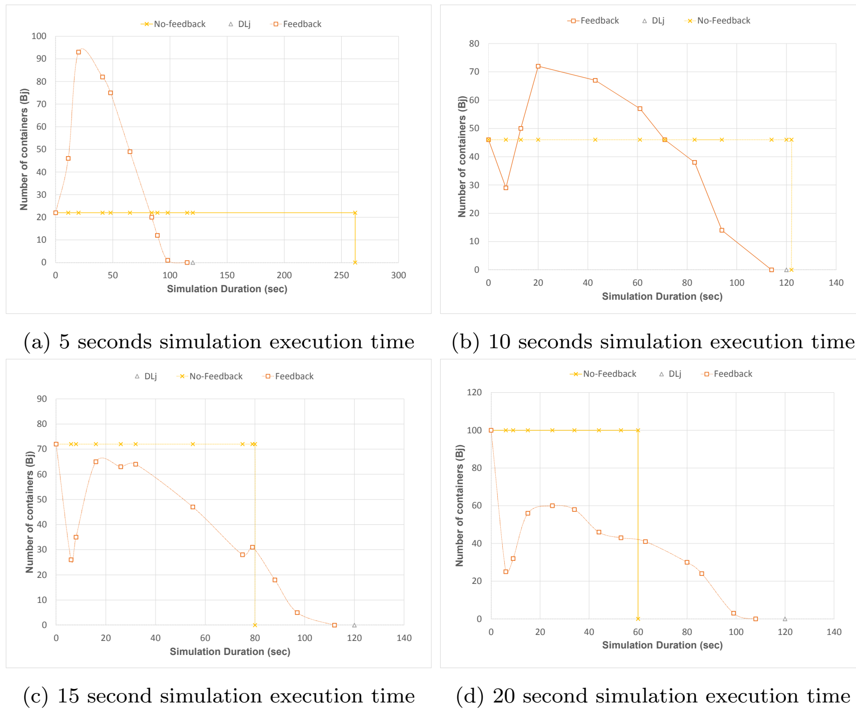
Another observation we make from the results is that a pessimistic execution time estimate (here 20s) results in more initial resource allocation. Resource allocation has its own cost. For example, we measured the cost of deployment of simulation from the private registry to the docker hosts for both of our use cases. For the image deployment of heater simulation of use case 1, it took 135.1 sec and for the SUMO simulation deployment of use case 2, it took 34.6 sec. These values are network-dependent but are incurred one-time per host which can be done as a setup process.

Since resource allocation incurs cost, an optimistic estimate is better because the system can adjust itself. However, if the estimate is too optimistic, the system may not be able to finish the job within the user-defined deadline.

Test 3 – Validating the Applicability of the Feedback-based Approach: The goal of applying Algorithm 1 is to spread the load per job over the deadline period so that multiple jobs can run in parallel while meeting their deadlines. In other words, the system does not schedule a job (and its tasks) immediately upon

a request but delays it such that the load is balanced yet ensuring that the deadline will be met.

Figure 6 compares a scenario where we do not apply the feedback-based approach and instead allocate resources based on a fixed expected execution time. Estimating an accurate execution time is not a trivial task and may not even be realistically feasible. The execution time does not just depend on the input parameter and hardware; it is also dependent on the performance interference due to other processes running on the shared resource. Too little a value, and we miss the deadline while too high value will result in wastage of resources. We observe from the results that the feedback based approach meets the deadline in all the cases while minimizing the resource consumption by releasing the containers if not needed.



(a) 5 seconds simulation execution time

(b) 10 seconds simulation execution time

(c) 15 second simulation execution time

(d) 20 second simulation execution time

Fig. 6: Comparison of Feedback vs No Feedback Approaches

Test 4 – Varying Host Overbooking Ratios: This set of experiments were performed to measure the capabilities of the system to handle multiple parallel requests made to the hosts with varying overbooking ratios, and study their performance while executing the containers. Table 4 shows the results of the experiments where we varied the overbooking ratio from 0.5 to 6. The specified deadline was 4 minutes and expected simulation time was 10 seconds.

Table 4: System Performance with Varying Host Overbooking Ratios

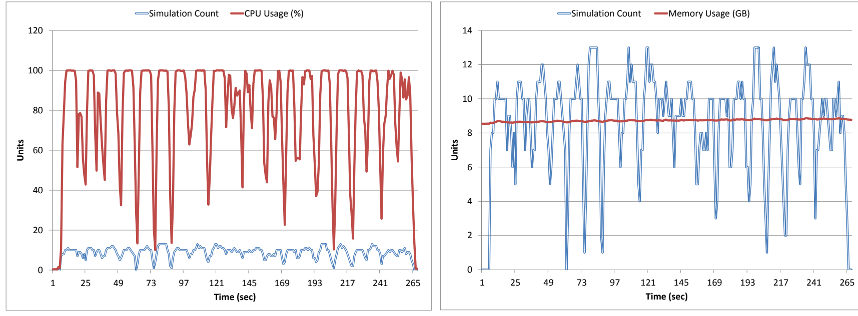
Overbooking Ratio	Max Container Count	Max Hosts Acquired	Simulation Duration (in sec)	Measured Execution Time per Sim (in sec)	Turnaround Time per Sim (in sec)	Measured Overhead(%)
0.5	29	5	239.4	9.31	10.48	12.57
1	41	4	233.6	9.53	11.52	20.88
2	59	3	231.3	15.62	18.59	19.01
4	114	3	213.3	28.72	32.76	14.07
6	160	3	Deadline Missed	41.62	46.9	12.67

We measure the container count and the actual 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 simulation duration 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, the actual simulation execution time per simulation and the corresponding system overhead. This overhead includes the performance interference overhead, resource contention and the time consumed in data transfer at different components of the SIMaaS workflow as shown in Figure 3.

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 remains at a reasonable level of less than 21% during the experiments.

Based on the experiments, we illustrate in Figures 7, the CPU utilization and memory utilization for use case 1. 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. The results were similar for use case 2.

Test 5 – Varying Number of Simulations: The purpose of these tests is to demonstrate the scalability of SIMaaS middleware with increasing number of simulations that are needed as the fidelity of statistical model checking increases. The tests were executed with a deadline of 600 seconds while other parameters were kept the same as in previous experiments. Table 5 shows the results, which illustrates that the system is able to scale to 5,000 simulations for a job without significant overhead.



(a) CPU Utilization Variations with Simulation Count (b) Memory Utilization Variations with Simulation Count

Fig. 7: Use Case 1: CPU Utilization Variations with Simulation Count

Table 5: System Performance with Varying Number of Simulations

Number of Simulations	Max Container Count	Max Hosts Acquired	Simulation Duration (in secs)	Measured Execution Time per Sim (in sec)	Turnaround Time per Sim (in ms)	Measured Overhead(%)
500	18	1	513.8	4.81	7.79	61.95
1000	33	2	547.1	5.26	11.45	117.68
2500	71	3	588.5	5.52	11.02	99.64
5000	137	6	591.4	5.49	10.72	95.26

Test 6 - Varying Simulation Execution Time: For these experiments, we vary the sampling rate parameter of use case 1 and use a deadline of 10 minutes to increase the simulation execution time of our simulation model. Table 6 measures and presents the simulation performance for varying execution time. We observe that the overhead reduces significantly as the duration of the container execution increases, which is attributed mainly to less percentage of time spent in scheduling and start up of containers.

Table 6: System Performance with Varying Simulation Duration

Sampling Rate (in ms)	Max Container Count	Max Hosts Acquired	Simulation Duration (in secs)	Measured Execution Time per Sim (in sec)	Turnaround Time per Sim (in ms)	Measured Overhead(%)
10	6	1	526.3	4.69	5.87	25.91
5	11	1	533.0	9.35	10.48	12.09
1	108	9	526.0	63.91	66.81	4.53

Test 7 – Scalability Test and Admission Control with Incoming Workload: These are scalability experiments with the workload described in Section 5.3

with 60 docker hosts and an incoming request flow generated using Poisson distribution. The system applied admission control and informed the users about the decision to accept the request. Table 7 summarizes the results for the tests.

Table 7: Scalability Test Summary

Number of Jobs	103
Test Duration	1h:47min:57s
Number of Simulations Performed	15873
Hosts Utilized	54 / 60
Jobs Rejected	1
Number of Jobs that Missed Deadline	1

We observed that one job with deadline of 618 seconds missed it by 1.39 seconds. This failure can be eliminated with stricter error estimation parameter (α), explained in Section 4.1. SIMaaS scaled to 54 virtualized hosts during the experiments. In future, we would like to experiment with a larger cluster to test the system's scalability limit.

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, or systems whose models result in different outcomes for different parameters.

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 application 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 suggest that an overbooking ratio of 2 and α value of 1.5 provided the best configuration to execute the simulations. However, these conclusions were based on the existing use cases 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.

– We have handled basic failures in our system where a container is scheduled again if it does not start, however we need advanced fault tolerance mechanism to handle failure of hosts and various SIMaaS components. Our prior work [6] has explored the use of VM-based fault tolerance, however, for the current work we will need container-based fault tolerance mechanisms.
– We did not consider a billing model for the end users in our work but such a consideration should be given to generate revenues for such a service and also so that the user does not abuse the system.
– As most of the cloud providers are rolling out Linux container-based application deployment, we need to design a hybrid cloud to leverage it.
– Currently our middleware architecture is realized as a centralized deployment in our small-scale private data center. In large data centers, we will require a distributed realization of the various entities of our middleware. This will give rise to a number of distributed systems issues, and addressing these form the dimensions of our future work.

All the scripts and source code, and experimental results of SIMaaS are available for download from `http://www.dre.vanderbilt.edu/~sshekhar/download/SIMaaS`.

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. An K, Shekhar S, Caglar F, Gokhale A, Sastry S (2014) A Cloud Middleware for Assuring Performance and High Availability of Soft Real-time

Applications. Elsevier Journal of Systems Architecture (JSA) 60(9):757–769, DOI http://dx.doi.org/10.1016/j.sysarc.2014.01.009
7. Atzori L, Iera A, Morabito G (2010) The Internet of Things: A Survey. Computer networks 54(15):2787–2805
8. 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`
9. Behrisch M, Bieker L, Erdmann J, Krajzewicz D (2011) SUMO-Simulation of Urban MObility-an Overview. In: SIMUL 2011, The Third International Conference on Advances in System Simulation, pp 55–60
10. Van den Bossche R, Vanmechelen K, Broeckhove J (2011) Cost-efficient Scheduling Heuristics for Deadline Constrained Workloads on Hybrid Clouds. In: Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, IEEE, pp 320–327
11. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Software: Practice and Experience 41(1):23–50
12. 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
13. Calheiros RN, Netto MA, De Rose CA, Buyya R (2013) EMUSIM: An Integrated Emulation and Simulation Environment for Modeling, Evaluation, and Validation of Performance of Cloud Computing Applications. Software: Practice and Experience 43(5):595–612
14. Casanova H, Giersch A, Legrand A, Quinson M, Suter F (2013) SimGrid: A Sustained Effort for the Versatile Simulation of Large-scale Distributed Systems. arXiv preprint arXiv:13091630
15. 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
16. Fujimoto RM (1990) Parallel Discrete Event Simulation. Communications of the ACM 33(10):30–53
17. Fujimoto RM, Malik AW, Park A (2010) Parallel and Distributed Simulation in the Cloud. SCS M&S Magazine 3:1–10
18. 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
19. García-Valls M, Cucinotta T, Lu C (2014) Challenges in Real-Time Virtualization and Predictable Cloud Computing. Journal of Systems Architecture
20. Haklay M, Weber P (2008) Openstreetmap: User-generated Street Maps. Pervasive Computing, IEEE 7(4):12–18

21. 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
22. Hellegouarch S (2007) CherryPy Essentials: Rapid Python Web Application Development. Packt Publishing Ltd
23. Kenyon C, et al (1996) Best-Fit Bin-Packing with Random Order. In: SODA, vol 96, pp 359–364
24. 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
25. Knuth DE (1969) The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Revised Edition
26. Lardieri P, Balasubramanian J, Schmidt DC, Thaker G, Gokhale A, Damiano T (2007) A Multi-layered Resource Management Framework for Dynamic Resource Management in Enterprise DRE Systems. Journal of Systems and Software: Special Issue on Dynamic Resource Management in Distributed Real-time Systems 80(7):984–996
27. 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
28. 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
29. 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
30. LXC (2014) Linux Container. URL https://linuxcontainers.org/, last accessed: 10/11/2014
31. 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
32. Mauch V, Kunze M, Hillenbrand M (2013) High Performance Cloud Computing. Future Generation Computer Systems 29(6):1408–1416
33. 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
34. 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
35. 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

36. Rasmussen CE, Williams CKI (2005) Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press
37. Shankaran N, Kinnebrew JS, Koutsoukas XD, Lu C, Schmidt DC, Biswas G (2009) An Integrated Planning and Adaptive Resource Management Architecture for Distributed Real-time Embedded Systems. Computers, IEEE Transactions on 58(11):1485–1499
38. Shipyard (2014) Shipyard Project. URL `http://shipyard-project.com/`, last accessed: 10/11/2014
39. 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`
40. 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
41. 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
42. 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
43. 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`
44. 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
45. 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