SOFTWARE: PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2017; **47**:505–521
Published online 27 June 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2422

ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers

Sareh Fotuhi Piraghaj*,†, Amir Vahid Dastjerdi, Rodrigo N. Calheiros and Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Melbourne, Victoria, Australia

SUMMARY

Containers are increasingly gaining popularity and becoming one of the major deployment models in cloud environments. To evaluate the performance of scheduling and allocation policies in containerized cloud data centers, there is a need for evaluation environments that support scalable and repeatable experiments. Simulation techniques provide repeatable and controllable environments, and hence, they serve as a powerful tool for such purpose. This paper introduces *ContainerCloudSim*, which provides support for modeling and simulation of containerized cloud computing environments. We developed a simulation architecture for containerized clouds and implemented it as an extension of CloudSim. We described a number of use cases to demonstrate how one can plug in and compare their container scheduling and provisioning policies in terms of energy efficiency and SLA compliance. Our system is highly scalable as it supports simulation of large number of containers, given that there are more containers than virtual machines in a data center. Copyright © 2016 John Wiley & Sons, Ltd.

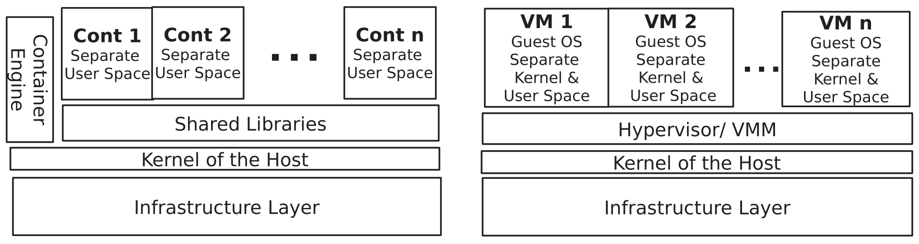
Received 3 March 2016; Accepted 26 May 2016

KEY WORDS: cloud computing; simulation; container as a service (CaaS); containerized clouds

1. INTRODUCTION

Because of the elasticity, availability, and scalability of its on-demand resources, cloud computing is being increasingly adopted by businesses, industries, and governments for hosting applications. In addition to traditional cloud services, namely, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), recently a new type of service—Containers as a Service (CaaS)—has been introduced. An example of container management system is Docker§ that allows developers to define containers for applications. Containers share the same kernel with the host; hence, they are defined as lightweight virtual environments compared to virtual machines (VMs) that provide a layer of isolation between workloads without the overhead of hypervisor-based virtualization. CaaS can lie between IaaS and PaaS; while IaaS provides virtualized compute resources and PaaS provides application specific runtime services, CaaS glues these two layers together by providing isolated environments for the deployed applications (or different modules of an application). As illustrated in Figure 1, CaaS services are usually provided on top of IaaS' virtual machines. CaaS providers, such as Google and AWS, argue that containers offer appropriate environment for semi-trusted workloads, while virtual machines provide another layer of security for untrusted workloads.

*Correspondence to: Sareh Fotuhi Piraghaj, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia.
†E-mail: sarehfotuhi@gmail.com
§Docker: https://www.docker.com/.


Container Engine
Cont 1 Separate User Space
Cont 2 Separate User Space
Cont n Separate User Space
Shared Libraries
Kernel of the Host
Infrastructure Layer
VM 1 Guest OS Separate Kernel & User Space
VM 2 Guest OS Separate Kernel & User Space
VM n Guest OS Separate Kernel & User Space
Hypervisor/ VMM
Kernel of the Host
Infrastructure Layer

(a) Operating System (OS) Level. (b) System Level.

Figure 1. OS level versus System level virtual environments.

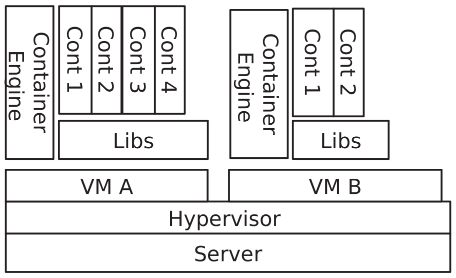


Figure 2. The virtual environment modeled in *ContainerCloudSim*.

Resource management policies to ensure Quality of Service (QoS), avoid energy wastage, and resource fragmentation are an integral part of cloud systems. Innovating and comparing resource management strategies require evaluation environments that facilitate the design of experiments while making them repeatable and accurate. Simulators are useful tools to build such evaluation environment in the cloud context [1]. They are particularly helpful at early stages of research to identify and eliminate ineffective polices or when accessing large scale distributed infrastructure is costly and not possible. Testing and evaluating resource management policies in the first verification stage in a production environment is both risky and costly. In this respect, a number of simulation tools are developed for evaluation of algorithms that are specifically designed for cloud computing environments. Although containers are going to be one of the dominant application deployment models in the cloud, most of the simulators consider VMs as the building blocks of the virtualized cloud data centers.

To the best of our knowledge, no simulators introduced modeling for containerized cloud environments. In this paper, we propose a simulation environment, named '*ContainerCloudSim*', for studying resource management techniques in CaaS environments. *ContainerCloudSim* is developed as an extension of the CloudSim simulation toolkit [2]. It provides an environment for evaluation of resource management techniques such as container scheduling, placement, and consolidation of containers. As depicted in Figure 2, *ContainerCloudSim* uniquely enables researchers to consider resource management techniques for both virtualization types including the Operating System level virtualization/containers (Figure 1(a)) and system level virtualization/VMs (Figure 1(b)) side by side. For the *Virtual Machine* type, applications execute inside virtual machines, and for the CaaS model, applications execute inside containers while the containers are placed in virtual machines. The proposed simulator models a container migration by stopping the container on the source host and starting it with a realistic delay on the destination host, which closely resembles current containerized environments. Moreover, *ContainerCloudSim* offers an environment to evaluate various (power-aware) resource management algorithms by providing diverse power models in a data center.

2. RELATED WORK

As we discussed earlier, simulation environments can speed up the development process of theoretical research by allowing repeatable experiments in a controllable environment [3]. Simulators enable the study of the effect of one parameter on the objective of the research while keeping the other parameters controlled, which might be difficult or sometimes impossible to achieve in a real world scenario. Considering the significant benefits of simulation for cloud computing environments, a number of simulators with various objectives were developed [4]. The simulators differ in considered performance metrics, supported applications, and whether they consider power consumption or not.

MDCSim [5] is a commercial comprehensive and scalable simulation toolbox that is used for in-depth analysis of multi-tier data centers. It models the underlying hardware characteristics of data center components and estimates the power consumption of data centers. Throughput and response time are considered as performance metrics, and the topology of the data center is supplied as a directed graph by the MDCSim network package. MDCSim helps cloud users to examine different resource configurations to improve the performance of web applications while keeping the power consumption low. Likewise, GDCSim [6], as a simulation tool, is especially developed to help service providers to test the impact of different data center physical designs and resource management algorithms on power consumption before deployment. GDCSim is extensible so that the user can add new models of power consumption, resource management, and cooling. Similar to GDSim[6] and MDCSim [5], *ContainerCloudSim* also enables users to have an estimate of the data center power consumption by using the available built-in or user-defined power models.

CloudSim is developed as an extensible cloud simulation toolkit that enables modeling and simulation of cloud systems and application provisioning environments [2]. This toolkit provides both system and behavior modeling of cloud computing components such as virtual machines (VMs), data centers, and users. It also enables the evaluation of resource provisioning policies in a cloud computing environment. The generic application provisioning techniques implemented in CloudSim can be extended easily with limited effort. It also supports modeling and simulation of both single and inter-networked clouds (federation of clouds) and exposes custom interfaces for implementing policies and VM provisioning techniques. In *ContainerCloudSim*, CloudSim is extended to enable modeling a containerized cloud environment that is not currently supported by CloudSim or any of its extensions.

Simulation Program for Elastic Cloud Infrastructures (SPECI) [7] is a simulation toolkit that focuses on scalable design of cloud data centers. In addition, it is capable of testing failure and recovery mechanisms. This enables exploring aspects of scalability along with performance properties of future data centers. The objective of SPECI is simulating the performance and behavior of data centers having the size and middleware design policy as input.

GroudSim [8] is a Java-based simulation toolkit especially designed for simulating scientific applications execution both on Grid and cloud infrastructures. GroudSim provides users with basic statistics and analysis after the simulation. It also supports modeling of computational and network components, job submissions, and file transfers. Similar to SPECI [7], failures in GroudSim can be modeled and integrated with background load and cost models. *ContainerCloudSim* can also be extended to incorporate the modeled application and machine failures.

Data center Simulator (DCSim) [9] is an extensible simulation framework developed aiming at investigating dynamic resource management techniques in Infrastructure as a Service cloud deployment model. The key features introduced in DCSim includes a multi-tier application model and the modeling of interactions and dependencies between virtual machines (VMs). VM replication is another feature available in DCSim and is utilized for handling increases in the workload.

GloudSim [10] is developed as a distributed cloud simulator based on the second version of the Google traces considering virtualization technology (VMs). GloudSim introduced three main features. The first feature is the ability to emulate resource utilization of reproduced jobs as closely as possible to the real values in the trace. The second feature is the simulator's ability to precisely emulate the different event types such as kill/evict based on the trace. Finally, the simulator can emulate more complex cases beyond the original trace investigating the challenges in resource management in cloud computing environment. GloudSim can also reproduce check-pointing/restart events considering the Google trace leveraging Berkeley Lab Checkpoint/Restart (BLCR) tool. Like GloudSim [10], *ContainerCloudSim* also provides the support for incorporating the cloud data centers' resource utilization traces. Currently, the workload models of PlanetLab [11] is utilized as the container's usage data.

iCanCloud [12] is a simulation tool that aims to predict the trade-offs between cost and performance of a set of applications executed in a cloud data center. iCanCloud can be used by different users including basic cloud users and distributed application developers. The simulation platform of iCanCloud provides a scalable, fast, and easy-to-use tool helping users to obtain results quickly considering their budget limits. iCanCloud is based on SIMCAN[‡] and provides a graphical user interface that enables users to execute the experiments. In addition, it models network communication between machines and supports parallel simulations; hence, an experiment can be executed across multiple machines.

In CloudAnalyst, Wickremasinghe *et al.* [13] extended CloudSim to enable applications workload description including the number of users, data centers, and cloud resources along with the location of both users and data centers. CloudAnalyst can be used by application developers or testers to determine the best strategic allocation of resources among the available cloud data centers. Data centers can be selected strategically considering the application workload and the available budget. Like iCanCloud [12], CloudAnalyst also provides a graphical interface that simplifies the process of building a number of simulation scenarios.

In TeachCloud [14], CloudSim is extended with a model for MapReduce application and an integrated comprehensive workload generator called *Rain*. It enables experiments with various cloud components including processing elements, storage, networking, and data centers. Like the two aforementioned simulators [12, 13], it also supports a graphical interface that enables building and implementing customized network topologies. It also provides a VL2 network topology model. TeachCloud, as a comprehensive and easy-to-use tool, can be utilized as a educational tool that allows students to conduct experiments in a cloud system. As a future work, we will provide a graphical interface for *ContainerCloudSim* that shows the utilization details of the modeled data centers components.

CDOSim [15] is developed extending CloudSim to model response times, SLA violations, and costs of a cloud deployment option (CDO). CDOSim is oriented towards the cloud user side instead of investigating the issues on the cloud provider side. The user behavior can be supplied through workload profiles extracted from production monitoring data. CDOSim can simulate any application as long as it can be modeled following the Knowledge Discovery Meta-Model (KDM) [16]. The notion of million instructions per second (MIPS) unit in CloudSim is refined to the mega integer plus instructions per second (MIPIPS) unit. CDOSim is integrated in the cloud migration framework CloudMIG [17] and is available as a plug-in for the CloudMIG Xpress tool. Contrary to CDOSim [15], *ContainerCloudSim* is mainly focused on the provider's side.

In NetworkCloudSim, Garg *et al.* [18] extended CloudSim to provide a scalable network and generalized application model. It supports applications with communicating elements including Message Passing Interface (MPI) and workflows. A network flow model is designed for cloud data centers, and bandwidth sharing is made possible. The extension is developed in such a way that users can modify the network topology simply by modifying a configuration file.

Contrary to NetworkCloudSim [18], GreenCloud [19] is developed as a packet level simulator on top of the NS2 simulator [20]. GreenCloud is specifically designed to investigate power management schemes to achieve an energy efficient data center. These schemes include both voltage and frequency scaling, and dynamic shutting down of network and compute components. It enables the capture of details of the energy consumption of data center's computing and network components. It also considers the packet-level communication patterns between network components. Like CloudSim, ContainerCloudSim can also be extended to incorporate network components and the communications between containers.

In summary, in comparison to our proposed *ContainerCloudSim*, existing simulators do not support modeling and simulation of containers in a cloud environment. They primarily focus on system

[‡]http://www.arcos.inf.uc3m.es/~simcan/.

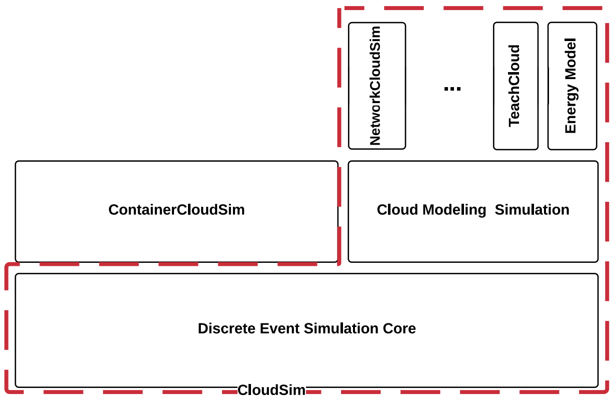


Figure 3. *ContainerCloudSim* relations to the CloudSim ecosystem. [Colour figure can be viewed at wileyonlinelibrary.com]

level virtualisation with virtual machine as the fundamental component [6, 8–10, 12–15, 18, 19, 21]. In comparison to other CloudSim extensions [13, 14, 18, 21], *ContainerCloudSim* has been developed on top of the CloudSim core as depicted in Figure 3.

3. CAAS MODELING REQUIREMENTS

Because Container as a Service (CaaS) is a newly introduced service in public cloud computing environments, there is still a lack of defined methods and standards that can efficiently tackle both application-level and infrastructure complexities. However, as container technologies mature and are broadly adopted, research in this topic will also emerge, proposing new algorithms and policies for containerized cloud environments. To reduce the development time of these new approaches, there is need for an environment that provides functionalities to enable robust experiments with various set-ups allowing development of best practices in a containerized cloud context. In this respect, we developed *ContainerCloudSim* that aims to provide support for modeling and simulation of containerized cloud computing environments including

- Management interfaces for containers, VMs, hosts, and data centers resources including CPU, memory, storage. Particularly, it should provide the fundamental functionalities such as provisioning of VMs to containers, dynamic monitoring of the system state, and controlling the application execution inside the containers.
- Functionalities that enable researchers to plug in and compare new container scheduling and provisioning policies. Container scheduling policies determine how resources are allocated to containers and virtual machines, and can be extended to allow evaluation new strategies.
- Investigation of energy efficient resource allocation ability of provisioning algorithms. The simulation environment should provide basic models and entities that can be utilized to evaluate the energy aware provisioning algorithms. To this end, container migration and consolidation have to be supported.
- Support for simulation scalability, as the number of container in a CaaS environment is much higher than the number of virtual machines in a data center.

4. SIMULATOR ARCHITECTURE

ContainerCloudSim follows the same layered architecture of CloudSim, with necessary modifications to introduce the concept of containers. In the proposed architecture of *ContainerCloudSim* (as depicted in Figure 4), CaaS consists of containerized cloud data centers, hosts, virtual machines, containers, and applications along with their workloads. For efficient management of CaaS, the architecture benefits from multiple layers:

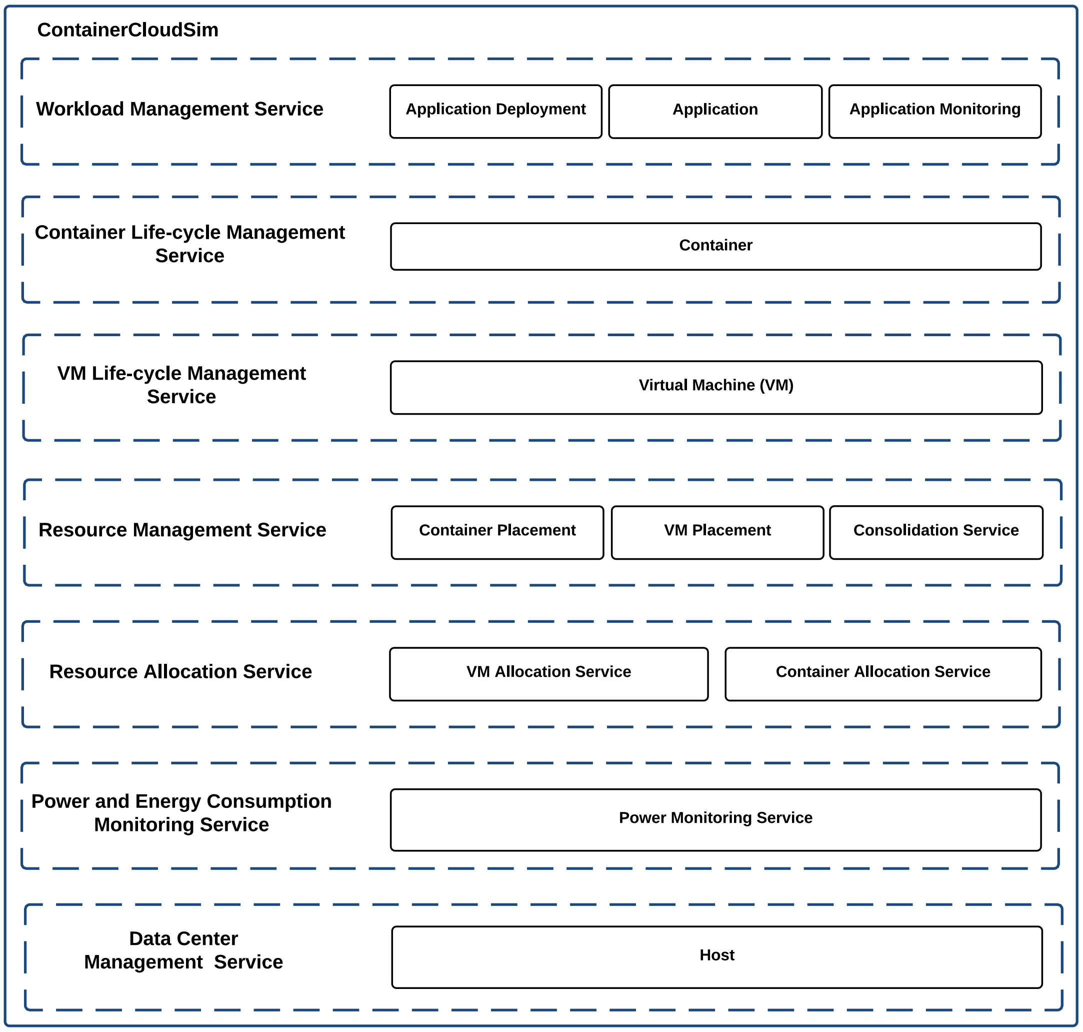

ContainerCloudSim
Workload Management Service
Application Deployment
Application
Application Monitoring
Container Life-cycle Management Service
Container
VM Life-cycle Management Service
Virtual Machine (VM)
Resource Management Service
Container Placement
VM Placement
Consolidation Service
Resource Allocation Service
VM Allocation Service
Container Allocation Service
Power and Energy Consumption Monitoring Service
Power Monitoring Service
Data Center Management Service
Host

Figure 4. *ContainerCloudSim* simulator architecture. [Colour figure can be viewed at wileyonlinelibrary.com]

Workload Management Service: This service takes care of clients' application registration, deployment, scheduling, application level performance, and health monitoring.

Container Life-cycle Management Service: This service is responsible for container life-cycle management. This includes creating containers and registering them in the system, starting, stopping, restarting, and migrating containers from a host to another host, or destroying the container. In addition, this component is responsible for managing the execution of tasks that are running inside the container and monitoring their resource utilization.

VM Life-cycle Management Service: This service is responsible for VM management and consists of VM creation, start, stop, destroy, migration and resource utilization monitoring.

Resource Management Service: This service manages the process of creating VMs/containers on hosts/VMs that satisfy their resource requirements and other placement constraints such as software environment. It consists of three main services:

- Container Placement Service: Containers are allocated to the VMs based on a Container allocation policy defined in this service.
- VM Placement Service: VMs are allocated to hosts considering a VM allocation policy that is defined in the VM placement service.
- Consolidation Service: This service minimizes resource fragmentation by consolidating containers to the least number of hosts.

Resource Allocation Service: This service manages the allocation of resources to VMs and containers. It consists of the following services:

- Container Allocation Service: This service is equipped with policies that determine how VM resources are allocated (scheduled) to containers.
- VM Allocation Service: This service is equipped with policies that determine how hosts' resources are allocated (scheduled) to VMs.

Power and Energy Consumption Monitoring Service: This services is responsible for measuring the power consumption of hosts in the data center and is equipped with the necessary power models.

Data Center Management Service: This services is responsible for managing data center resources, powering on and off the hosts, and monitoring the utilization of resources.

5. DESIGN AND IMPLEMENTATION

For implementing the aforementioned functionalities, CloudSim Discrete Event simulator Core is used to provide basic discrete event simulation functionalities and modeling of basic cloud computing elements. Because CloudSim entities and components communicate through message passing operations, the core layer is responsible for managing events and handling interactions between components. The main classes of *ContainerCloudSim* are depicted in Figure 5. In this section, we go through the details of these classes. *ContainerCloudSim* implementation is constituted of two main parts *simulated elements* and *simulated services*. The simulated elements include the following:

- Datacenter: The hardware layer of the cloud services is modeled through the Datacenter class.
- Host: The Host class represents physical computing resources (servers). Their configurations are defined as processing capability that is expressed in MIPS (million instructions per second), memory, and storage.
- VM: This class models a VM. Virtual Machines are managed and hosted by a Host. Attributes of VM are memory, processor, and its storage size.
- Container: This class models a Container that is hosted by a VM. Attributes of Containers are accessible memory, processor, and storage size.
- Cloudlet: The Cloudlet class models applications hosted in a container in cloud data centers. Cloudlet length is defined as Million Instructions (MI), and it has functionalities of its predecessor in CloudSim package including StartTime and status of execution (such as CANCEL, PAUSED, and RESUMED).

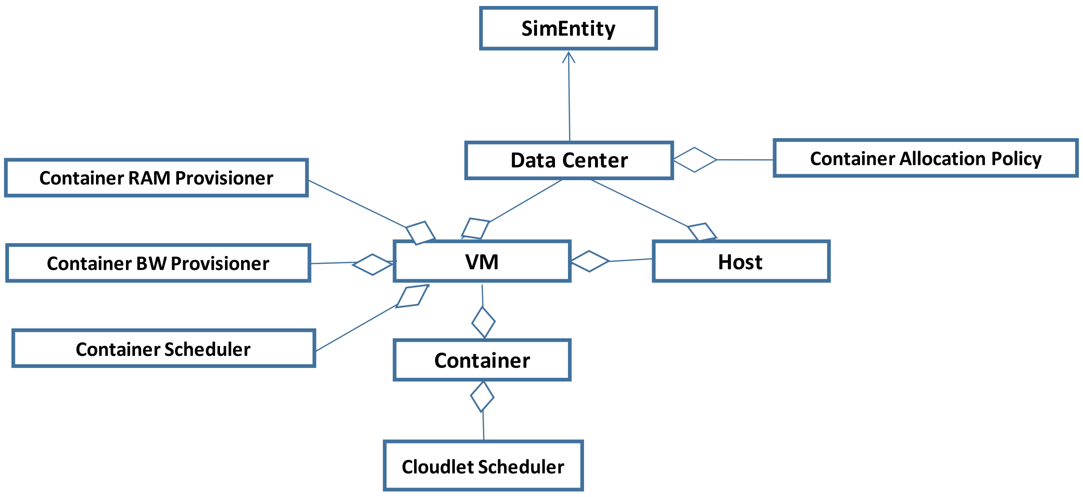
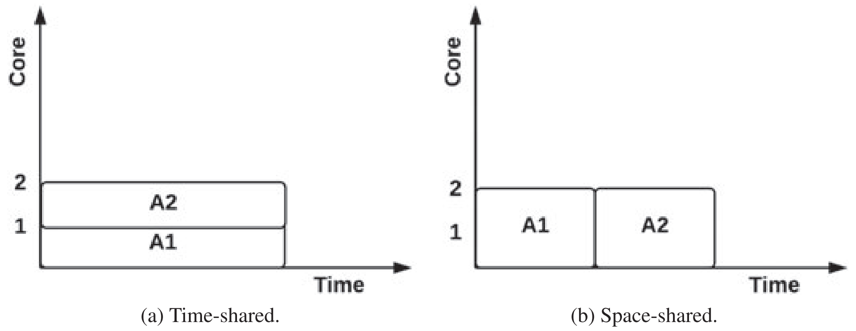


Figure 5. *ContainerCloudSim* class diagram. [Colour figure can be viewed at wileyonlinelibrary.com]

In addition, simulated services available in *ContainerCloudSim* are as follows:

- VM Provisioning: The VM provisioning policy, which assigns CPU cores from the host to VMs, is considered as a field of the Host class. Similar to CloudSim, the Host component implements the interfaces that provide modeling and simulation of classes that implement CPU cores management. For example, a VM can have dedicated cores assigned to it (pinning of cores to VM) or can share cores with other VMs in the host.
- Container Provisioning: The simulator provides container provisioning at two levels: VM level and container level. At the VM level, the amount of VM's total processing power that is assigned to each container is specified. Whereas at the container level, the container can assign a fixed amount of resources to each of the application services that are hosted on it. To enable compatibility with CloudSim, a task unit is considered as a finer abstraction of an application service that is hosted in the container. Time-shared and space-shared provisioning policies are implemented for both levels in the current version of the *ContainerCloudSim* (as depicted in Figure 6). In addition,ContainerRamProvisioner is an abstract class that represents the provisioning policy utilized for allocating the virtual machine's memory to containers. A container can be hosted on a VM only if the ContainerRamProvisioner component assures that the VM has the needed amount of free memory. If the memory requested by the container is beyond the VM's available free memory, the ContainerRamProvisioner rejects the request. For provisioning the bandwidth, the abstract class named ContainerBwProvisioner models the policy for provisioning of bandwidth of the containers. The role of this component is handling network bandwidth allocation to a set of competing containers. This class can be extended to contain new policies to include the requirements of various applications.

  Figure 6 illustrates a simple provisioning scenario. In this figure, containers $A1$ and $A2$ are hosted on a host with 2 cores. In the space-shared scenario, only one of the two $A1$ and $A2$ containers can run at a given instance of time. Therefore, $A2$ can only be assigned the core when $A1$ finishes its execution. In this scenario, each container requires 2 cores for its execution. However, in the time-shared scenario, each container receives a time slice on each processing core and each component receives a variable amount of the processing power during its execution. The available amount of processing power for each container can be estimated through the calculation of the number of active components that are hosted on each VM. The provisioning policy is defined by ContainerScheduler, which is an abstract class and is implemented by a VM component. More application-specific processor sharing policies can be implemented by overriding the functionalities of this class.
- CloudletScheduler: The same relationship between container and VM is held between applications (called Cloudlets) and containers. The CloudletScheduler abstract class can be extended to implement different algorithms to identify the share of processing power among Cloudlets that are running in a container. Both types of provisioning policies are included in the *ContainerCloudSim* package, namely, time-shared (ContainerCloudletSchedulerTimeShared) and space-shared (ContainerCloudetSchedulerSpaceShared) policies.



(a) Time-shared. (b) Space-shared.

Figure 6. Time-shared and space-shared provisioning concepts for containers A1 and A2 running on a VM.

- CloudInformationService: The CloudInformationService (CIS) class provides resource registration, indexing, and discovering capabilities.
- ContainerAllocationPolicy: This abstract class represents a placement policy that is utilized for allocating containers to VMs. The chief functionality of the ContainerAllocationPolicy is to select the available VM in a data center that meets the container's deployment requirements including the container's required memory, storage, and availability. Different placement policies, with different objectives, are created by extending this class.
- VmAllocationPolicy: In addition to allocating VMs to hosts, this abstract class implements the *optimizeAllocation* method that defines the consolidation policies in container and VM levels.
- Workload Management: Highly variable workloads is one of the main characteristics of cloud applications. In this respect, *ContainerCloudSim* also supports the modeling of dynamic workload patterns of cloud applications in a CaaS environment. We leveraged the existing Utilization Model in CloudSim to determine resource requirements on container-level. The Utilization Model is an abstract class and its getUtilization() method can be overridden by simulator users to obtain various workload patterns. The getUtilization() method input is the simulation time, and its output is the percentage of the required computational resource of each Cloudlet.
- Data Center Power Consumption: To manage power consumption per host basis, the PowerModel class is included. It can be extended (by overriding the getPower() method) for simulating custom power consumption model of a host. getPower() input parameter is the host's current utilization metric while its output is the power consumption value. By using this capability, *ContainerCloudSim* users are able to design and evaluate energy-conscious provisioning algorithms that demand real-time monitoring of power utilization of cloud system components. The total energy consumption can also be reported at the end of the simulation.

5.1. Discrete-event simulation dynamics

The simulated processing of task units is managed inside the containers executing the tasks. In this respect, at every simulation step, the task execution progress is updated. Figure 7 depicts the sequence diagram of the updating process. At each simulation time step, the method updateVMsProcessing() of the Datacenter class is called. updateVMsProcessing() method accepts the current simulation time as its input parameter type. It then calls a method (updateContainersProcessing()) on each host to instruct them to update the processing on each of their VMs. The process is recursively repeated for each VM to update their container processing and for each container to update the application processing. The method at the container level returns the earliest completion time of jobs running on it. At VM level, the smallest completion time among all containers is returned to the host. Finally, at host level, the smallest completion time among all VMs is returned to Datacenter. The earliest time value returned to the Datacenter class is used to set the time in which the whole process will be repeated. An event is then scheduled in the simulation core for the calculated time, which dictates the next simulation step, and therefore progresses the simulation.

6. USE CASES AND PERFORMANCE EVALUATION

To demonstrate the capabilities of *ContainerCloudSim* for evaluating resource management policies, we present three use cases including the container overbooking, container consolidation, and container placement. Further, we evaluate the *ContainerCloudSim* in terms of its scalability and container start-up delay modeling.

All the use cases leverage the architecture depicted in Figure 8. In this architecture, the VMM deployed on top of physical servers sends the data including the status of the host along with the list of containers that are required to be migrated to the consolidation manager. The consolidation manager, which is deployed on a separate machine, decides about the new placement of containers and sends requests to provision resources to the destination host.

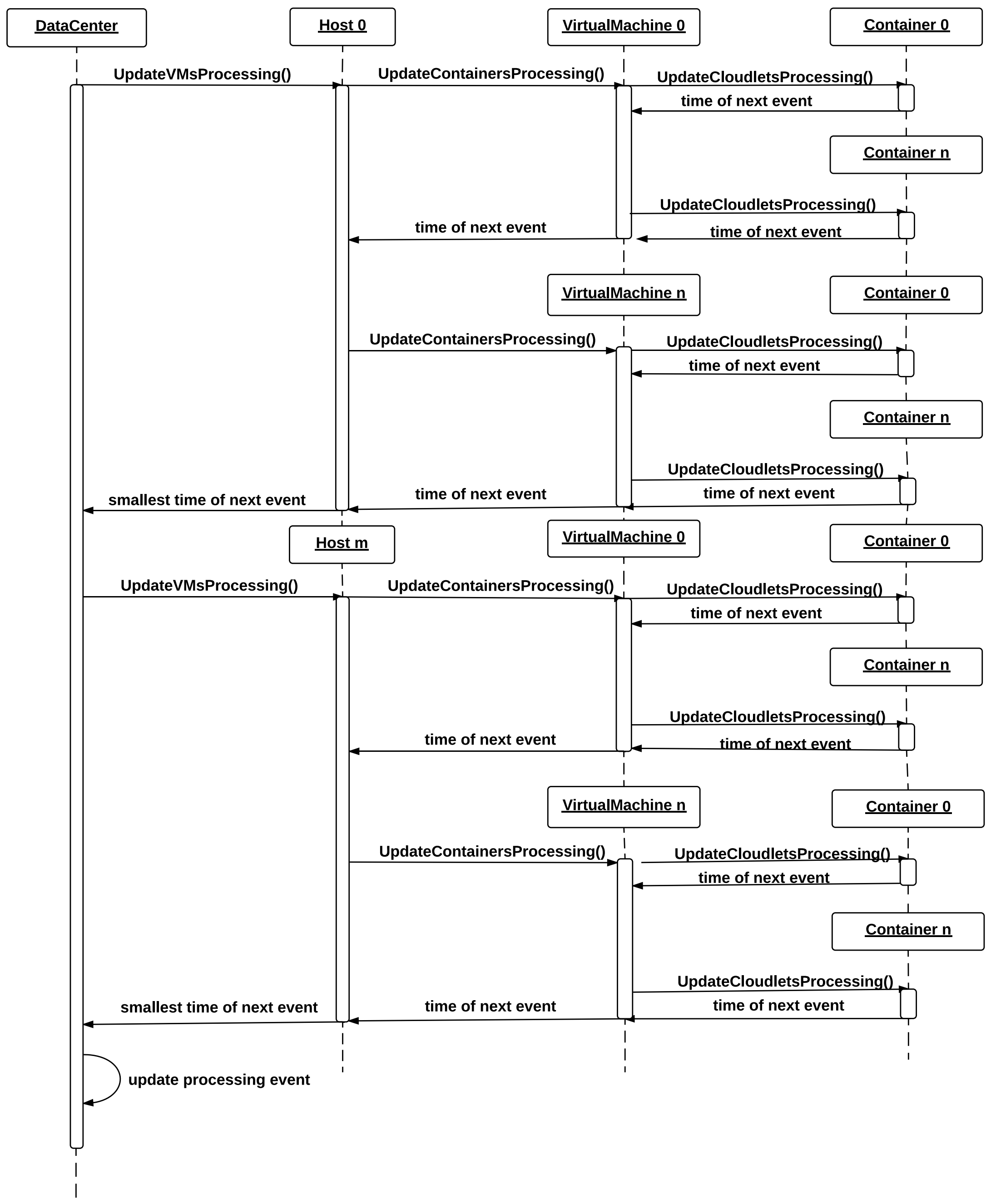


Figure 7. Data center internal processing sequence diagram.

6.1. *Use Case 1: container overbooking*

Cloud users tend to overestimate the container size they require so that they can avoid the risk of facing less capacity than the actually required by the application. The user's overestimation provides opportunity for the cloud providers to include an overbooking strategy [22] in their admission control system to accept new users based on the anticipated resource utilization rather than the requested amount. Overbooking strategies manage the trade-off between maximizing resource utilization and minimizing performance degradation and SLA violation. *ContainerCloudSim* is capable of overbooking containers by allocating resources for a specific percentile of the workload.

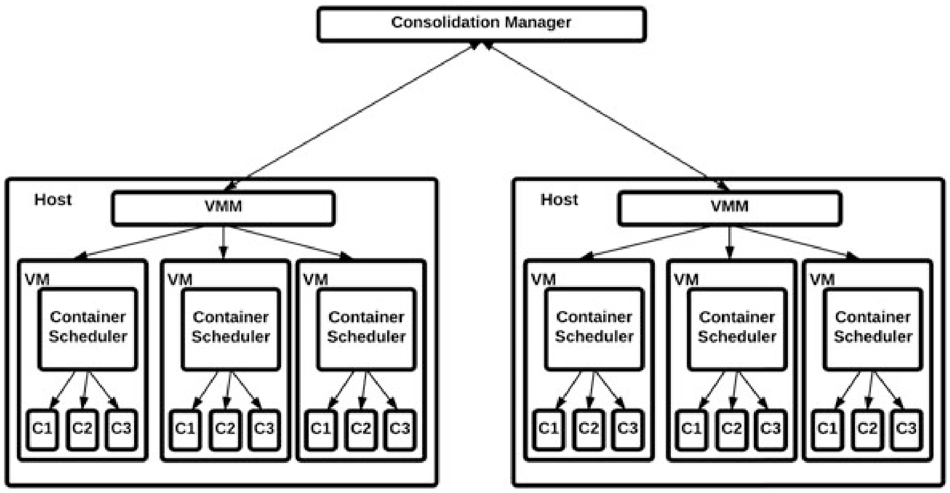


Figure 8. A common architecture for the studied use cases: VMM sends the data including the status of the host along with the list of the containers to migrate to the consolidation manager. The consolidation manager decides about the destination of containers and sends requests to provision resources to the selected destination.

In this case study, to demonstrate this capacity of the simulator, we designed a couple of experiments to investigate the impact of container overbooking. In the designed experiments, containers are placed on virtual machines according to a pre-defined percentile of their workload, which varied from 10 to 90. The workload traces are derived from PlanetLab [11] and are used as the containers' CPU utilization. These traces contain 10 days of the workload of randomly selected sources from the testbed that were collected between March and April 2011 [23]. Containers are placed on the VMs using First Fit algorithm.

In these experiments, we also utilized the consolidation capability of the simulator. In this respect, the migration process is triggered if the host status is identified as over-utilized/underutilized. A simple static threshold-based algorithm is utilized for this purpose. Hosts with less than 70% or more than 80% CPU utilization are considered overloaded or under-loaded, respectively. When the migration is triggered because of an overloaded host, the containers with the highest CPU utilization are chosen to migrate. The simulation set up including the configurations of the servers, containers, and virtual machines are all shown in Table I.

The output of the simulation is depicted in Figure 9(a) and shows that the number of successfully allocated containers decreases as the percentile increases. The higher percentile results in a smaller number of containers accommodated on each VM. The same trend exists when the number of container migrations is considered (Figure 9(b)). The volatility of the workload is the key factor that affects the percentile value. Thus, more volatile workloads would show more difference in the simulation results.

6.2. Use Case 2: container consolidation

Container consolidation is a promising approach to decrease energy consumption. *ContainerCloudSim* supports this by modeling container migrations aiming at consolidating containers to a smaller number of hosts. In *ContainerCloudSim*, a migration is triggered either because a host is overloaded or under-loaded. To this end, a number of containers should be selected for the migration list in order to rectify the situation. Utilizing *ContainerCloudSim*, researchers are able to study various selection algorithms and investigate the efficiency of their proposed selection policies in terms of the desired metrics including the data center energy consumption, container migration rate, and SLA violations.

The aim of this case study is utilizing *ContainerCloudSim* to investigate the container selection algorithm effect on the efficiency of the consolidation process of the containers. The same set up depicted in Table I is considered; however, in order to evaluate the algorithms in a larger scale, a

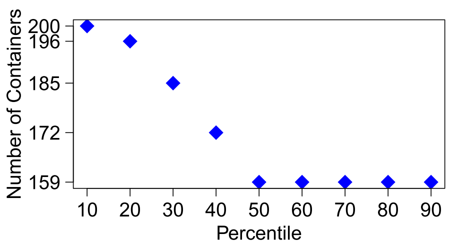
Table I. Configuration of the server, VMs, and containers.

Server Configurations and power models (20 Servers)

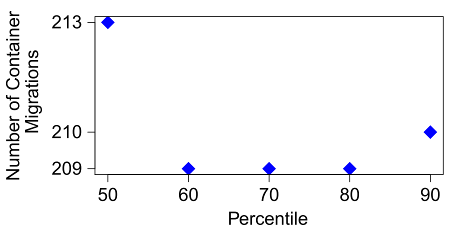
Server type #	CPU [3 GHz] (mapped on 37274 MIPS Per core)	Memory (GB)	$P^{\text{idle}}(Watt)$	P^{max} (Watt)	Population
# 1	8 cores	128	93	135	20

Container and VM Types (200 Containers and 20 VMs in total)

Container Type #	CPU MIPS (1 core)	Memory (GB)	Population	VM Type #	CPU [1.5 GHz] (mapped on 18636 MIPS Per core)	Memory(GB)	Population
# 1	4658	128	66	# 1	2 cores	1	5
# 2	9320	256	67	# 2	4 cores	2	5
# 3	18636	512	67	# 3	1 core	4	5
				# 4	8 cores	8	5

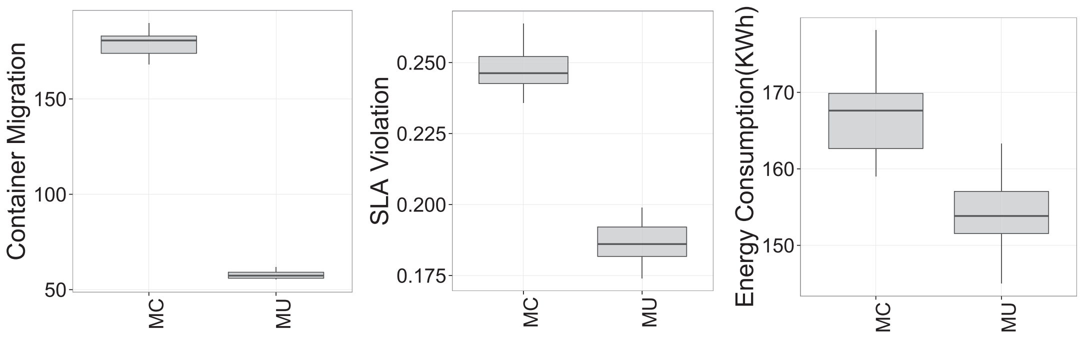


(a) The number of containers which are successfully allocated to the VMs considering each predefined percentiles of the workload.



(b) The number of container migrations happened during the experiments.

Figure 9. Impact of container's overbooking on the number of successfully allocated containers along with the number of container migrations happened for the experiments with the same number of allocated containers. [Colour figure can be viewed at wileyonlinelibrary.com]



(a) (b) (c)

Figure 10. Impact of container selection algorithm on the container migration rate (per 5 minute), SLA violations, and the total data center energy consumption.

larger number of elements are considered in this case study: the number of containers, VMs, and servers set to 4002, 1000, and 700, respectively. Under-load and overload thresholds are fixed as in the previous use case and are 70% and 80%, respectively. Containers are placed utilizing the First-Fit algorithm. The destination host is also selected based on the First Fit policy.

The two studied algorithms for containers selection are the 'MaxUsage' and the 'MostCorrelated' algorithms. The 'MaxUsage' algorithm selects the container that has the biggest CPU utilization while the 'MostCorrelated' algorithm chooses the container whose load is the most correlated with the server that is hosting it. Each experiment is repeated 30 times, and results are compared and depicted in Figure 10. The power consumption of the data center at time t $(P_{dc}(t))$ is calculated as $P_{dc}(t) = \sum_{i=1}^{N_S} P_i(t)$, where N_S is the number of servers and P_it corresponds to the power consumption of $server_i$ at time t. CPU utilization is applied for estimating the power consumption of each server as CPU is the dominant component in a server's power consumption [24]. The linear power model $P_i(t) = P_i^{idle} + \left(P_i^{max} - P_i^{idle}\right) * U_{i,t}$ is applied for calculating the servers power consumption, where P_i^{idle} and P_i^{max} are the idle and maximum power utilization of the server, respectively, and $U_{i,t}$ corresponds to the CPU utilization of server i at time t.

The SLA in this experiment is considered violated if the virtual machine on which the container is hosted does not receive the required amount of CPU that it requested. Therefore, the SLA metric is defined as the fraction of the difference between the requested and the allocated amount of CPU for each VM. The SLA metric is shown in Equation 1 [23] in which N_s, N_vm, and N_c are the number of servers, VMs and containers, respectively. In this equation, $\text{CPU}_{r(vm_{j,i},t_p)}$ and $\text{CPU}_{a(vm_{j,i},t_p)}$ correspond to the requested and the allocated CPU amount to vm_j on server i at time t_p.

$$SLA = \sum_{i=1}^{N_\text{s}} \sum_{j=1}^{N_\text{vm}} \sum_{p=1}^{N_\text{c}} \frac{\text{CPU}_r(vm_{j,i}, t_p) - \text{CPU}_a(vm_{j,i}, t_p)}{\text{CPU}_r(vm_{j,i}, t_p)} \tag{1}$$

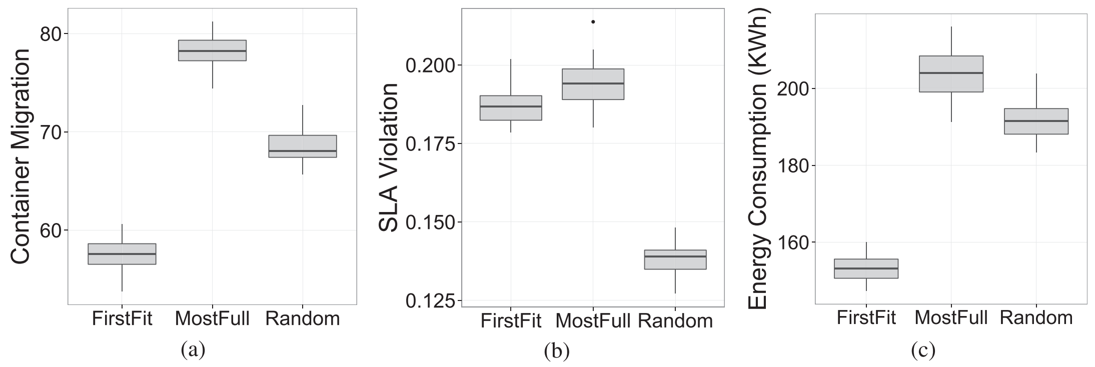


Figure 11. Impact of initial container placement algorithm on the container migration rate (per 5 minute), SLA violations, and data center energy consumption.

As shown in Figure 10, adding the containers with the maximum CPU utilization to the migration list results in less container migrations, energy consumption, and SLA violations and thus should be the preferred policy to be utilized by CaaS providers.

6.3. *Use case 3: container placement policies*

Various mapping scenarios between containers and virtual machines result in different resource utilization patterns. Researchers can utilize *ContainerCloudSim* to study various container to VM mapping algorithms. Therefore, in this case study, we demonstrate how *ContainerCloudSim* is used to investigate the effect of container placement algorithms on the number of container migrations, data center total power consumption, and resulting SLA violations. The same set up as of the *Use Case 2* is applied. The three different placement policies are evaluated: FirstFit, MostFull, and random. As depicted in Figure 11, the MostFull placement algorithm, which packs containers on the most full virtual machine in terms of the CPU utilization, results in a higher container migration rate. Consequently, the aforementioned algorithm results in higher violations and energy consumption. In contrast, FirstFit results in less number of migrations and energy consumption and thus should be the preferred policy to be utilized if the goal of the provider is to reduce energy consumption.

6.4. *Container and VM start-up delays*

An important operation in cloud computing environments is instantiation of virtual machines. This time is non-negligible and can impact performance of applications running on clouds. Virtual machine start-up delay of virtual machines was previously studied by Mao et. al [25]. Based on this study, the current version of the simulator includes a static delay of 100 seconds for every virtual machine.

Containers startup delay is also important, because live migration of containers is not applicable in real-world scenarios. Therefore, the container migration is performed through shutting down the container on the source host as soon as the same container is started on the destination host. Likewise VMs, container startup delay can be set as a constant (ConstantsEx.Container_STARTTUP_DELAY) in the current version of *ContainerCloudSim*.

The Docker containers startup delay has been recently studied for running one to 100 Ubuntu containers on top of one of the Amazon EC2 instances (c4.4xlarge)[§]. Each container runs the server *Uptime* command. This command is used for identifying how long a system has been running. The storage backend devicemapper is utilized along with an Ubuntu Linux image. In order to have a better understanding of the startup delay, we also followed the same set up. However, we increased the number of containers simultaneously executed from 100 to 5000 (adding one container at a time). The experiment is conducted for other selected Amazon EC2 instance types. As Figure 12 shows,

[§]Docker Performance Tests: http://www.draconyx.net/articles/some-docker-performance-tests.html.

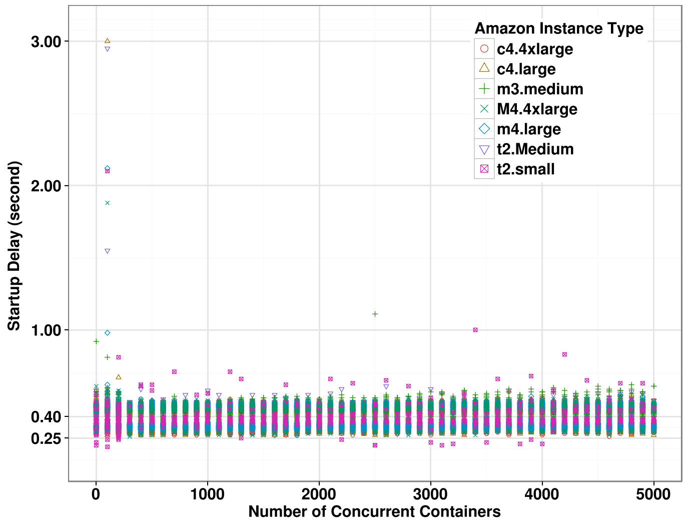
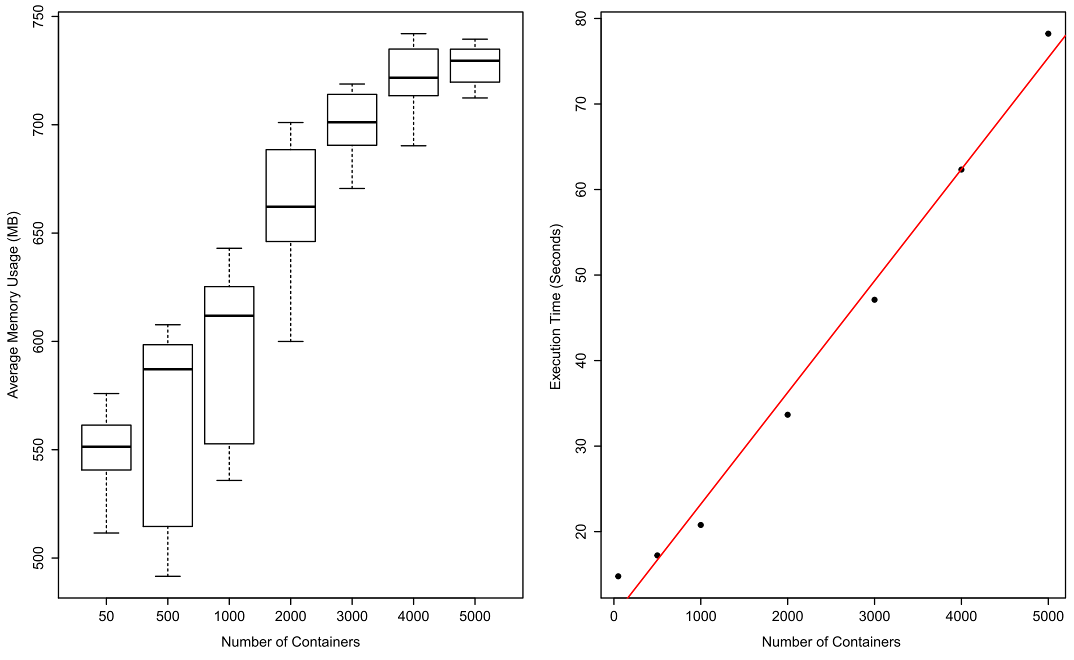


Figure 12. The container start up delay for running 1 to 5000 concurrent containers in each of the studied Amazon EC2 instances. [Colour figure can be viewed at wileyonlinelibrary.com]



(a) (b)

Figure 13. Impact of increasing the number of containers on the average memory usage and the execution time of the simulator. [Colour figure can be viewed at wileyonlinelibrary.com]

the start up delays varies between 0.2 and 0.5 seconds for most of the cases. For the current version of the simulator, we utilized the average container startup delay for all studied instance types, which is equal to 0.4 seconds.

6.5. *Simulation scalability*

As a containerized cloud simulator, *ContainerCloudSim* scales with minimal memory overhead and execution time. In order to investigate the scalability of the developed simulator, we ran the same experiment set up as Case 2 considering the MostFull algorithm as the container initial placement policy. The same experiment is repeated for various number of containers ranging from 50 to 5000.

In order to have a fair comparison, the number of available hosts and VMs are considered constant and equal to 700 and 1000, respectively. For each experiment, the execution time of the simulation, which is defined as the time that it takes for the simulation to finish, and the memory utilization of the Java program (simulation) are depicted in Figure 13. The experiment results show (Figure 13(a)) that the memory overhead for running 5000 containers is less than 200MB on average. Considering the execution time as it is depicted in Figure 13(b), with every 1000 containers added the simulation time increases by almost 20 seconds only. It shows that *ContainerCloudSim* is scalable enough to enable simulation experiments on the scale expected in the context of Container as a Service.

7. CONCLUSIONS AND FUTURE WORK

We discussed modeling and simulation of containerized computing environments as they are currently one of the dominant application deployment models in clouds. We proposed the *ContainerCloudSim* simulator architecture and implemented it as an extension of CloudSim. We carried out three use cases and demonstrated effectiveness of the *ContainerCloudSim* for evaluating resource management techniques in containerized cloud environments. Moreover, scalability of simulation is verified, and the approach for modeling container migration is validated in a real environment. Our experiment results demonstrated that *ContainerCloudSim* is capable of supporting simulations on the scale expected in the context of CaaS. We also believe that the availability of our simulator will energize research in CaaS policies.

As future work, we plan to extend *ContainerCloudSim* to visualize spatio temporal behavior of nodes, VMs, and containers in a data center. We also plan to model the connectivity between containers for supporting modeling of application such as Web applications and MapReduce-like computing environments.

SOFTWARE AVAILABILITY

The *ContainerCloudSim* code is available for download along with CloudSim software from website: http://www.cloudbus.org/cloudsim.

REFERENCES

1. Zhao W, Peng Y, Xie F, Dai Z. Modeling and simulation of cloud computing: A review. *Proceedings of the 2012 IEEE Asia Pacific Congress on Cloud Computing (APCloudCC)*, Shenzhen, 2012; 20–24.
2. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience* 2011; **41**(1):23–50.
3. Stojmenovic I. Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions. *IEEE Communications Magazine* 2008; **46**(12):102–107.
4. Ettikyala K, Devi YR. Article: A study on cloud simulation tools. *International Journal of Computer Applications* 2015; **115**(14):18–21.
5. Lim SH, Sharma B, Nam G, Kim EK, Das CR. MDCSim: a multi-tier data center simulation, platform. *Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops*, New Orleans, LA, 2009; 1–9.
6. Gupta SKS, Banerjee A, Abbasi Z, Varsamopoulos G, Jonas M, Ferguson J, Gilbert RR, Mukherjee T. GDCSim: a simulator for green data center design and analysis. *ACM Transactions on Modeling and Computer Simulation* 2014; **24**(1):3:1–3:27.
7. Sriram I. SPECI, a simulation tool exploring cloud-scale data centres. In *Cloud Computing*, vol. 5931, Jaatun M, Zhao G, Rong C (eds)., Lecture Notes in Computer Science. Springer-Verlag: Berlin, Heidelberg, 2009; 381–392.
8. Ostermann S, Plankensteiner K, Prodan R, Fahringer T. GroudSim: an event-based simulation framework for computational grids and clouds. *Proceedings of the 2010 Conference on Parallel Processing*, Euro-Par 2010, Springer-Verlag, Berlin, Heidelberg, 2011; 305–313.
9. Tighe M, Keller G, Bauer M, Lutfiyya H. DCSim: a data centre simulation tool for evaluating dynamic virtualized resource management. *Proceedings of the 2012 8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualiztion Management (SVM)*, Las Vegas, NV, 2012; 385–392.
10. Di S, Cappello F. GloudSim: google trace based cloud simulator with virtual machines. *Software: Practice and Experience* 2015; **45**(11):1571–1590.
11. Park K, Pai VS. CoMon: a mostly-scalable monitoring system for planetlab. *SIGOPS Operating Systems Review* 2006; **40**(1):65–74.

12. Nez A, Vzquez-Poletti J, Caminero A, Casta G, Carretero J, Llorente I. iCanCloud: a flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing* 2012; **10**(1):185–209.
13. Wickremasinghe B, Calheiros R, Buyya R. CloudAnalyst: a cloudsim-based visual modeller for analysing cloud computing environments and applications. *Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, Perth, WA, 2010; 446–452.
14. Jararweh Y, Alshara Z, Jarrah M, Kharbutli M, Alsaleh MN. TeachCloud: a cloud computing educational toolkit. *International Journal of Cloud Computing* 2013; **2**(2-3):237–257. PMID: 55269.
15. Fittkau F, Frey S, Hasselbring W. CDOSim: simulating cloud deployment options for software migration support. *2012 IEEE 6th International Workshop on the Proceedings of the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, Trnto, 2012; 37–46.
16. Prez-Castillo R, de Guzmn IGR, Piattini M. Knowledge discovery metamodel-iso/iec 19506: A standard to modernize legacy systems. *Computer Standards and Interfaces* 2011; **33**(6):519–532.
17. Frey S, Hasselbring W. Model-based migration of legacy software systems into the cloud: The CloudMIG approach. *Softwaretechnik-Trends* 2010; **30**(2):84–85.
18. Garg SK, Buyya R. NetworkCloudSim: modelling parallel applications in cloud simulations. *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*, Victoria, NSW, 2011; 105–113.
19. Kliazovich D, Bouvry P, Khan S. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing* 2012; **62**(3):1263–1283.
20. *The network simulator - ns-2*. Available at: http://www.isi.edu/nsnam/ns/ [Accessed on 30 November 2015].
21. Calheiros RN, Netto MA, De Rose CA, Buyya R. Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. *Software: Practice and Experience* 2013; **43**(5):595–612.
22. Tomas L, Klein C, Tordsson J, Hernandez-Rodriguez F. The straw that broke the camel's back: Safe cloud over-booking with application brownout. *Proceedings of the 2014 International Conference on Cloud and Autonomic Computing (ICCAC)*, London, 2014; 151–160.
23. Beloglazov A, Buyya R. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computing : Practice and Experience* 2012; **24**(13):1397–1420.
24. Blackburn M, Grid G (eds.) *Five Ways to Reduce Data Center Server Power Consumption*. The Green Grid Administration: Beaverton, Oregon, USA, 2008.
25. Mao M, Humphrey M. A performance study on the vm startup time in the cloud. *Proceedings of the 2012 IEEE Fifth International Conference on Cloud Computing*, CLOUD '12, IEEE Computer Society: Washington, DC, USA, 2012; 423–430.