CloudSim 7G: An Integrated Toolkit for Modeling and Simulation of Future Generation Cloud Computing Environments

Remo Andreoli[a,b,*], Jie Zhao[a], Tommaso Cucinotta[b] and Rajkumar Buyya[a]

[a]CLOUDS Lab, School of Computing and Information Systems, University of Melbourne, Melbourne, Australia
[b]ReTiS Lab, TECIP Institute, Sant'Anna School of Advanced Studies, Pisa, Italy

ARTICLE INFO

Keywords:
CloudSim; CloudSim 7G; Cloud Computing; Resource Management; Simulation

ABSTRACT

Cloud Computing has established itself as an efficient and cost-effective paradigm for the execution of web-based applications, and scientific workloads, that need elasticity and on-demand scalability capabilities. However, the evaluation of novel resource provisioning and management techniques is a major challenge due to the complexity of large-scale data centers. Therefore, Cloud simulators are an essential tool for academic and industrial researchers, to investigate on the effectiveness of novel algorithms and mechanisms in large-scale scenarios. This paper proposes CloudSim 7G, the seventh generation of CloudSim, which features a re-engineered and generalized internal architecture to facilitate the integration of multiple CloudSim extensions within the same simulated environment. As part of new design, we created a set of new generic classes to abstract common functionalities, and carried out an extensive refactoring and refinement of the codebase. The outcome is the removal of a significant amount of lines of code without loss of functionality, significantly better performance in both run-time and total memory allocated (up to 20% less heap memory allocated), as well as increased flexibility, ease-of-use, and extensibility of the framework. These improvements benefit not only CloudSim developers but also researchers and practitioners using the framework for modeling and simulating next-generation cloud computing environments.

1. Introduction

Over the past decade, Cloud Computing has evolved rapidly, becoming the dominant model for modern computing. Compared to previous paradigms, the characterizing aspects of Cloud Computing are the illusion of infinite resources available 24/7 on-demand over the network, and of infinitely scalable applications. This is possible thanks to virtualization technologies, which allow the sharing of physical machines, storage, and networking devices among multiple customers and organizations. A plethora of service providers are involved in the provisioning of Cloud services, including Cloud and Network Service providers, to enable the wide range of applications being built by customers. The goal of the cloud provider is to ensure a satisfying experience for the customer, usually in terms of performance, reliability, security, and cost, without compromising profit.

It is difficult and costly to evaluate policies for Cloud provisioning, workload management, and resource handling in a real environment since a cloud infrastructure is a large and complex system comprising interconnected geo-located datacenters. In this context, simulation toolkits [32, 31, 4] mitigate the issue and play an important role in research communities for testing and evaluating complex applications and novel resource management strategies through inexpensive and repeatable experiments. Furthermore, simulations provide a controlled environment to test the performance of resource provisioning policies and to easily reproduce the results.

CloudSim [8] has been a forefront simulation toolkit, and the de-facto standard, for evaluating resource management techniques in Cloud Computing environments, thanks to its ease-of-use and extensibility. The first iteration of CloudSim offered a machine virtualization layer to simulate Virtual Machines (VMs) and test techniques such as provisioning, scheduling, and consolidation. Thousands of contributions in the field of Cloud orchestration and resource management use CloudSim [3, 41, 51], as demonstrated by the 6400+ citations to the seminal paper. CloudSim is a completely customizable tool: all its components and related interactions can be extended. For instance, Cloud researchers can develop a custom scheduling policy, or embed power-awareness in their VM instances. As the number and type of services offered by Cloud service providers have increased, in parallel, simulation toolkits have also evolved. As a result, CloudSim sports a rich ecosystem of extensions, from now on referred to as "CloudSim modules", to model and simulate all sorts of resource management challenges in the Cloud context. The rapid growth in Cloud adoption not only relies on machine virtualization as its main feature, but also on many fundamental features such as Software Defined Networking (SDN), Network Function Virtualization (NFV), container-based virtualization, and serverless application execution models. The CloudSim research

*Corresponding author
Email address: remo.andreoli@santannapisa.it (R. Andreoli)
ORCID(s):

community developed new modules such as NetworkCloudSim [19], CloudSimSDN [44, 45], WorkflowSim [10], and ContainerCloudSim [37], among others, to accommodate these advancements.

Since its inception, the CloudSim ecosystem has received contributions from diverse researchers and developers with varying skill levels and coding styles. As highlighted by other authors [47, 43], these contributions often fail to fully leverage the toolkit's extensibility and are typically developed as independent packages, significantly hampering the reuse of these extensions. This has led to a degree of fragmentation within the ecosystem, making it difficult for CloudSim users to seamlessly integrate multiple modules into a single simulated scenario. The culprit is CloudSim's lack of generalized interfaces for implementing the simulated entities and their interactions. A module developer creating an entirely new component had no easy way to maintain compatibility with components from other, unrelated CloudSim modules: developers have often been forced to copy-paste existing components and slightly modify their behaviors, to the detriment of the extensibility capabilities offered by Object-Oriented (OO) programming. For instance, ContainerCloudSim provided a collection of container scheduling and allocation policies which were the copy-pasted version of the VM-based ones. This is because VMs and containers have been portrayed as completely different components, despite performing the same task (i.e., executing user workloads). Another example: ContainerCloudSim provided a collection of host selection policies for placement purposes, whereas the power-aware package offered a collection of VM selection policies for migration purposes. The policies are fundamentally the same in both modules (i.e., select an entity from a list of candidates), but CloudSim does not provide a standard way to interface with them, forcing redundant code paths: one to handle migration, and one to handle placement. Previous versions of CloudSim have been released with a bundle of these independently packaged modules with no interoperability guarantees, contributing to some difficulty in the readability of the codebase.

1.1. Contributions

This paper proposes the seventh generation of the CloudSim toolkit, shortened to CloudSim 7G, the biggest re-engineering of the codebase to date. The major contributions of our proposal, in terms of enhanced functionalities and extensibility features, are:

1. A set of new generalized interfaces to standardize the definition, configuration, and creation of new components and their interactions. This enables the integration of several CloudSim modules, which were previously available independently and often had compatibility issues, within the same simulated scenario.
2. A re-engineered, refactored, and refined version of various modules from CloudSim 6G, for a total of more than 13,000 lines of code removed. In particular: i) The networking module of CloudSim, "NetworkCloudSim", has been rewritten almost completely; and ii) the container module, "ContainerCloudSim", has been heavily refactored to reduce its codebase.
3. The ability to simulate scenarios with nested virtualization: containers within VMs, VMs within containers, or even VMs within VMs, are all possible now. This allows a more accurate simulation of the typical configuration of a real-world cloud infrastructure: an example in the state-of-the-art are Kata Containers[1], where VMs are deployed within Kubernetes to improve isolation among them [52].
4. Significant performance improvement in simulation run-time and memory usage compared to previous CloudSim iterations, thanks to the various optimizations that have been carried out during the refactoring and refinement process.

The pre-packaged modules within CloudSim have been consolidated into a *base layer* to support the development of multi-module extensions. Therefore, the innovative capabilities of CloudSim 7G open up new opportunities for simulating next-generation Cloud Computing environments, allowing researchers to experiment with hybrid scenarios using various computing paradigms (i.e., Cloud Computing, Edge computing, Serverless computing, etc.). This ability for different modules to coexist and interact was previously not possible; To this end, significant attention has been given to minimizing the effort required for module developers to restore compatibility of their contributions with CloudSim 7G.

1.2. Paper Organisation

The rest of the paper is organized as follows: Section 2 recalls the evolution of the official CloudSim codebase, as released by researchers of the CLOUDS Lab, in conjunction with the paradigm shifts in Cloud Computing. Section 3 briefly presents a collection of external modules not developed within the CLOUDS Lab, as well as a subset of CloudSim's competitors in the literature. Section 4 presents: i) the proposed architectural change to the codebase of CloudSim so to allow for multi-extension simulations; ii) the essential refactoring, refinement and optimization steps performed to construct the new base layer of CloudSim; and iii) the guidelines to update an old module to CloudSim 7G. Section 5 presents a performance comparison between CloudSim 6G and CloudSim 7G in terms of run-time and total memory allocated, showcasing the improvements of CloudSim 7G in large simulations thanks to the massive optimization the codebase has undergone. Section 6 proposes a simple case study that uses multiple modules simultaneously, demonstrating the flexibility of the novel design. Finally,

[1] See: https://katacontainers.io/.

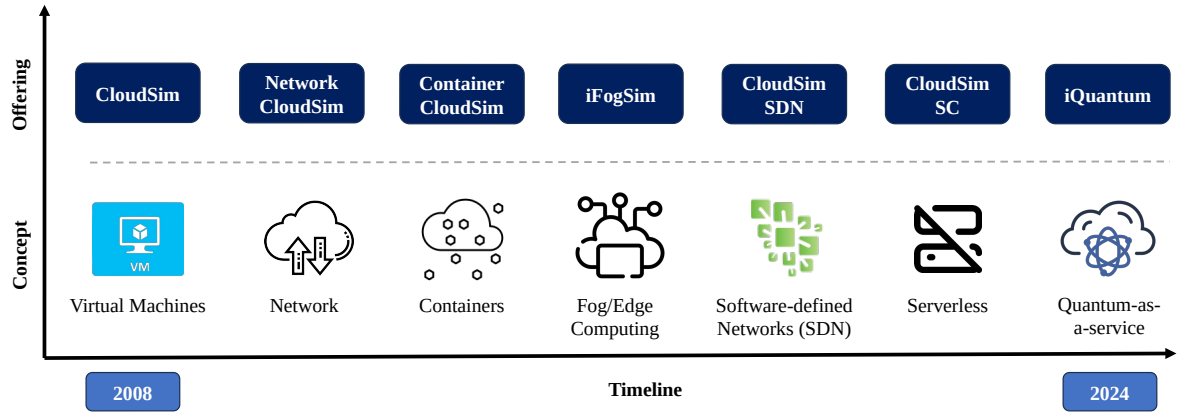


Figure 1: The evolution of CloudSim

Section 7 concludes the paper with final remarks and a discussion on future research directions for modeling and simulating future generation Cloud Computing environments.

2. CloudSim Through the Evolution of Cloud Computing and Related Paradigms

In the landscape of cloud simulators available in the literature [32, 31, 4], CloudSim, one of the first simulators specialized in evaluating resource management techniques for Cloud infrastructures, has established itself as the de-facto choice for the research community. Every simulated scenario follows these steps: the service broker submits an inventory to the datacenter, which comprises the virtual machines (VMs), physical hosts, and a list of ephemeral activities to be performed. The latter abstracts the concept of a request submitted to a Cloud service and in CloudSim jargon they are called cloudlets. The datacenter deploys the submitted inventory using a variety of resource provisioning methods and scheduling policies, both at the VM and cloudlet levels. The simulation terminates when all the submitted activities have been executed. The physical hosts and VMs are specified in terms of number of processing elements, available RAM, and network bandwidth. Each processing element has a specific processing strength measured in millions of instructions per second (MIPS), and the hardware resources allocated to each VM are limited by the capabilities of the physical host.

A cloudlet specifies an execution length, in terms of millions of instructions (MI), and the number of required processing elements to be provisioned. The actual execution time is determined by the capabilities of the underlying VM, the number of co-hosted activities on it, and the cloudlet scheduling policy. Out-of-the-box CloudSim supports two scheduling policies: space-shared, where only one cloudlet at a time can execute (others will be on a waiting list), and time-shared, where the processing strength is shared among cloudlets running simultaneously. More specifically, for time-shared scheduling, the start time of a cloudlet corresponds to the submission time, since there is no queuing, and the estimated finish time solely depends on the current processing capacity. For space-shared scheduling, the estimated start time depends on the cloudlet's position in the waiting list, whereas the current processing capacity of the VM is constant since always only one cloudlet at times is executing. Refer to the seminal paper at [8] for a more in-depth description of the simulated entities and their interactions. The power of CloudSim comes from the framework's extensibility, which allows researchers to customize existing policies, or create new ones, at different levels of the infrastructure.

2.1. CloudSim Official Releases

This section recalls the major characteristics of the milestones in the evolution of CloudSim, depicted in Figure 1, as officially released by researchers at the CLOUDS Lab. Notice that some of the described modules have been integrated into the base layer of CloudSim 7G.

CloudSim originated as a derivative of GridSim [6], enhancing its core functionalities with a new discrete-event management framework that more effectively represents the dynamic nature of cloud environments. The first iteration of CloudSim [7, 8] modeled the basic functionalities of machine virtualization described above, simple network behaviors in the form of a constant propagation delay calculated using a latency matrix, and a power module for investigating energy-aware VM consolidation policies. One of the shortcomings of the initial version of CloudSim is the absence of a proper network model. This limitation precludes the modeling of realistic scenarios with datacenter network topologies [26], which is crucial for a range of Cloud application domains, including scientific, big-data, and High-Performance Computing (HPC) workloads. In these contexts, applications are typically composed of several communicating tasks that need to be distributed across a cluster of interconnected physical machines. NetworkCloudSim [19] addressed such limitations with the introduction of a network flow model that simulates an internal network made of interconnected switches and physical machines. Consequently,

NetworkCloudSim also introduced a generalized application model for simulating complex workflow applications using a message-passing paradigm for communications. A "networked" cloudlet is structured as a sequence of stages, where each stage represents either a computational activity, like a "traditional" cloudlet, or a data transmission dependency (i.e., send/receive data). The communication between cloudlets that are part of the same workflow application but reside on different physical hosts is managed through a series of simulated switches.

The next evolution step came with the advent of Containers-as-a-Service (CaaS) offerings in the Cloud market. Containers represent a progression towards lightweight application management [34], hence it became evident the need for simulating such a novel virtualization paradigm. To this end, ContainerCloudSim [37] enriched CloudSim with new features for the evaluation of allocation, migration, and scheduling policies in containerized environments. In practice, ContainerCloudSim adds an additional scheduling layer to the simulated infrastructure, so that cloudlets are deployed within containers that are placed inside VMs.

The widespread adoption of virtualization technologies eventually reached the telecommunications field with the rising interest in Network Function Virtualization (NFV). Together with Software-Defined Network (SDN) technologies, they enable the deployment of network services, such as proxy and firewall ones, on commodity hardware instead of proprietary, expensive hardware appliances, much like virtualization is employed in cloud infrastructures to drive higher capacity utilization and reduce cost. CloudSimSDN [44, 45] enabled the evaluation of resource management strategies in infrastructures with SDN functionalities, such as dynamic network configuration and programmable controllers. In addition, CloudSimSDN supports the allocation, migration, autoscaling and service chaining for NFV. The simulation framework is based on the Open Source NFV Management and Orchestration (MANO) architecture [15].

The advent of the Internet-of-things (IoT) paradigm created the need for pushing computation and storage away from centralized data centers, and towards the "edge" of the network, to support the surge of big data. In this regard, iFogSim [20] enabled the simulation of IoT devices connected to Edge and Fog computing environments. The latest version [28] supports also the creation of complex microservice applications, as well as the simulation of service migrations for different mobility models of IoT devices and distributed cluster formation among Edge/Fog nodes of different hierarchical tiers.

The subsequent step in the history of CloudSim was supporting the evaluation of cloud-native applications [17]. In this context, Serverless computing gained much popularity thanks to the simplification it brings to the whole process of acquiring and managing cloud resources. The cloud provider takes on full responsibility for all operational tasks for provisioning and allocating the required resources to running applications, scheduling the applications on the infrastructure, and scaling the allocated resources to adapt to traffic changes. This enables developers to concentrate solely on the development of their cloud-native applications without the burden of server administration. CloudSimSC [30] is a toolkit for modeling serverless computing environments. This contribution offers a generalized architecture for function execution, scheduling, and load balancing, as well as resource scaling and monitoring metrics in terms of application performance, system throughput, and underlying resource consumption.

The latest trends in Cloud research are investigating hybrid quantum computing environments for solving computationally intractable problems. In this regard, iQuantum [33] extends the base CloudSim for the evaluation of scenarios with the presence of quantum resources. The module extracts the features of quantum circuits and then models them as workload entities to be executed on quantum datacenters.

3. Related Work

The CloudSim toolkit boasts a rich ecosystem of modules developed by researchers and institutions not affiliated with the CLOUDS Lab. These external, or "third-party", contributions are orthogonal to the evolution of CloudSim described in the previous subsection: sometimes an external contribution provides an alternative implementation of a milestone module or a direct improvement to it. Finally, we discuss about alternative Cloud simulators found in the literature.

3.1. CloudSim-driven External / Third-Party Modules

There are a plethora of external modules in the literature, those presented in this paper are summarized in Table 1 (first portion). One notable contribution is WorkflowSim [10] from the University of Southern California, an alternative to NetworkCloudSim which focuses on the simulation of the scheduling, clustering, and provisioning of large-scale scientific workflow workloads. A scientific workflow is expressed as a set of tasks that are interdependent in terms of data or control flow. In practice, the workloads are represented using the DAX format of PegasusWMS [13]. Compared to NetworkCloudSim, WorkflowSim adds an additional layer of workflow management to CloudSim to evaluate workflow optimization techniques such as task clustering [49]. Moreover, WorkflowSim supports the generation and monitoring of failures at run-time. A more recent contribution in this context is CloudSim4WDf [16] from University of Sfax. This extension examines the financial impact of dynamic changes of workflow. Application are represented using Business Process Modeling Notation (BPMN) to facilitate the modeling of workload structures changes at run-time. In comparison, WorkflowSim only supports the rigid structure imposed by the DAX file.

Name	Based on	Use-case
WorkflowSim [10]	CloudSim 3G	Clustering and failure modeling for scientific workflows
CloudSim4DWf [16]	CloudSim 4G	Dynamic workflows
CloudSimDisk [27]	CloudSim 3G	Disk operations modeling
CloudSimSFC [48]	CloudSim 3G	NFV
[40]	CloudSim 5G	Container orchestration
ACE [21]	CloudSim4G	Resiliency in Cloud
IoTSim [53]	CloudSim 4G	IoT for big data processing
edgeCloudSim [46]	CloudSim 4G	Mobility in Edge Computing
IoTSimEdge [22]	CloudSim 5G	IoT and Edge Computing
CloudSimPlus [43]	CloudSim 3G	Re-engineering of CloudSim's core simulation framework
CloudnetSim [11]	OMNeT++	Advanced CPU Scheduling
CloudnetSim++ [29]	OMNeT++	Energy consumption
Simcan2Cloud [9]	OMNeT++	Machine Virtualization
GreenCloud [23]	NS2	Energy consumption
DSLab IaaS [47]	DSLab	Machine virtualization
DSLab FaaS [42]	DSLab	Serverless Computing
FaaS-Sim [39]	SimPy	Serverless in Cloud-Edge scenarios

Table 1
A subset of the landscape of external CloudSim modules and alternate simulators.

CloudSimEx is a set of tools featuring disk operations, the simulation of MapReduce clusters [12], web sessions, and probabilistic cloudlet arrivals, among others. CloudSimDisk [27] from Luleå University of Technology is an alternative to the simple disk features of CloudSimEx that focuses on modeling the energy consumption of storage systems within a cloud infrastructure. Compared to CloudSimEx, CloudSimDisk provides a model for describing the characteristics of a Hard Disk Drive (HDD), an integration with the power modeling package of CloudSim, and a collection of algorithms for energy-aware storage management. CloudSimSFC [48], developed by Beihang University and based on CloudSimEx, is an alternative to CloudSimSDN that enhances the simulation of performance fluctuations in service chaining within Multi-domain Service Networks.

A contribution from German University in Cairo [40] provides an improved version of ContainerCloudSim that supports simulation scenarios with both VMs and containers, and dynamic arrival of cloudlets and workflow applications. However, the authors re-implemented each feature from scratch. CloudSim 7G supersedes this contribution since it allows to use VMs, containers (ContainerCloudSim), workflows (NetworkCloudSim, but also WorkflowSim), and specify a dynamic cloudlet arrival (CloudSimEx) in the same scenario without "yet another" independently-packaged re-implementation of such features.

FTCloudSim [54] from Beijing University of Posts and Telecommunications extends CloudSim to evaluate performance in the event of fault and recovery events using checkpointing [24]. A related work is ACE [21] from Western University Ontario, which investigates the resiliency of cloud infrastructure by enhancing CloudSim with support to availability-aware placement policies. ACE allows to inject failures at VM level and recover from them through a series of recovery and repair policies. Compared to FTCloudSim, ACE also explores replication as a fault-tolerant technique.

There is a significant collection of CloudSim modules focusing on different aspects of fog and edge computing. IoTSim [53] from Australian National University focuses on the simulation of IoT devices using the MapReduce model for big data processing. EdgeCloudSim [46] from Boğaziçi University is an alternative to iFogSim for modeling edge infrastructures. Compared to the first version of iFogSim, EdgeCloudSim considers mobility and implements a dynamic network communication model. IoTSimEdge [22] from Newcastle University focuses on composing IoT applications as microservices, offers enhanced mobility features compared to EdgeCloudSim, and examines also energy consumption implications.

A notable mention goes to CloudSimPlus [43] from Instituto Federal de Educação Ciência e Tecnologia do Tocantins, which is proposed as an alternative to the "base" CloudSim. The authors present a full rewrite of CloudSim3G to reduce code duplication and increase code reuse by improving several engineering aspects of CloudSim. The goal of CloudSimPlus is in common with our work, however, the latter suffers from two shortcomings: i) it is a fork of a very old version of CloudSim, and ii) it is not compatible with current CloudSim modules. Therefore, CloudSimPlus cannot support the rich ecosystem of contributions developed for CloudSim thus far. On the contrary, CloudSim 7G is a re-engineering attempt that stays as faithful as possible to previous CloudSim versions.

In conclusion, there is a vast array of modules that leverage the ease-of-use and extensibility of CloudSim to provide support for several computing paradigms. However, we observed significant overlap among these contributions (and particularly in the context of Edge computing), where each one is an ex-novo extension to CloudSim. This indicates that

CloudSim 7G's generalized interfaces could have facilitated building them on top of one another, rather than independently packaged modules, creating a more cohesive and readable framework.

3.2. Other Simulators

While many contributions to the simulation and modeling of Cloud Computing are CloudSim modules, the literature also presents several alternative approaches. CloudSim remains one of the most comprehensive simulators to date, largely thanks to the extensive array of modules for experimenting with the full spectrum of Cloud-related techniques and paradigms, making it unmatched by any other Cloud simulator in terms of features. By contrast, related works tend to offer distinct advantages tailored to specific scenarios.

As highlighted by a recent survey on the topic [31], most alternative cloud simulators in the literature are extensions of existing commercial network simulators. A subset of these contributions is described in what follows and summarized in Table 1 (second portion). CloudNetSim [11] was an extension of OMNeT++ [50], a commercial but open-source network simulator, to support the simulation of CPU scheduling within physical hosts and VMs, including hypervisor-level and guest OS-level hierarchical scheduling. It focused on the simulation of the Completely Fair Scheduler (CFS) default scheduler in the Linux kernel when used to schedule KVM VMs onto over-provisioned physical hosts, as well as to schedule tasks within a Linux guest VM. CloudnetSim++ [29] was another extension of OMNet++, developed independently from CloudNetSim, designed to study energy consumption at different components of geographically distributed cloud datacenters. These two platforms exploited OMNeT++ to offer accurate simulation of the TCP/IP network protocol, but in different contexts: scheduling algorithms and energy consumption, respectively. Simcan2Cloud [9] is one of the latest Cloud simulators based on OMNet++, which focuses in providing a highly-detailed simulated Cloud infrastructure. It models innovative aspects, including VM rental extensions, different queues for managing diverse access types, and prioritized resource allocation.

GreenCloud [23] was an extension of NS2, one of the most popular open-source network simulators, for evaluating energy-aware scheduling algorithms in cloud environments. It offered detailed modeling of energy consumption for hardware resources, including networking appliances such as switches and links, with differing compute and communication capacities, and SLA requirements. All these approaches have the advantage that they inherit packet-level accuracy from the underlying network simulator, but at the cost of long simulation times due to the large amount of "small" individual simulated components. Additionally, they lack support for simulating per-VM resource allocation and placement strategies, unlike a "general-purpose" simulator such as CloudSim.

Regarding ex-novo simulators, the authors in [47] propose DSLab IaaS, a novel simulator for IaaS infrastructures based on a general-purpose software framework for simulating distributed systems. Much like base CloudSim, DSLab IaaS's focus is on traditional machine virtualization. The simulator's standout feature is its ability to model multiple concurrently operating VM schedulers within the same host. The same authors also proposes DSLab FaaS [42] for the simulation of the Function-as-a-Service paradigm. However, there is no evidence that the two simulators can be used together, even though they originate from the same underlying framework. The primary drawback of both works is that are written in Rust, a programming language with a much smaller community of developers than Java or C, due to the much steep learning curve. On the contrary, much of CloudSim's popularity is to be attributed to the ease of use, whose only requirement is basic knowledge of the Java programming language.

FaaS-Sim is a Python-based, trace-driven stochastic simulation framework designed to evaluate placement and scaling decisions in serverless computing platforms. Given a network topology, a software architecture spanning the edge-cloud continuum and workload traces, FaaS-Sim provides estimates of function execution times and resource usage. Compared to CloudSimSC, it provides a more accurate resource model validated with real-world traces in comparison to NS2; Compared to FaaS-Sim, the focus is less tailored towards edge computing environments.

In conclusion, CloudSim has established itself as the tool of choice for researchers in this landscape thanks to: i) its ease of use, requiring only beginner-level Java knowledge and minimal expertise in specific domains, such as networking or serverless computing; ii) its abstraction level, which is well-suited for simulating key characteristics of Cloud scenarios without having to perform a fine-grained packet-level, hop-by-hop, simulation in a distributed worldwide network, as it would be needed with approaches based on network simulators; and iii) its wide array of extensions covering any Cloud-related computing paradigm.

4. CloudSim 7G

In CloudSim 7G, the codebase has undergone a massive refactoring and refinement process to accommodate the generalized interfaces that enables the coexistence of multiple modules within the same simulated scenario. As a positive side-effect, the refactoring and refinement process allowed us to modernize and optimize the original codebase, improving the simulation performance, as well as its readability, usability, and flexibility. See Section 5 for a performance evaluation between CloudSim 6G and CloudSim 7G. Feature-wise, CloudSim 7G contains refined versions of many spin-off contributions previously bundled with CloudSim as independent packages. These contributions include containers, geo-located services, web load balancing, and network flow modeling.

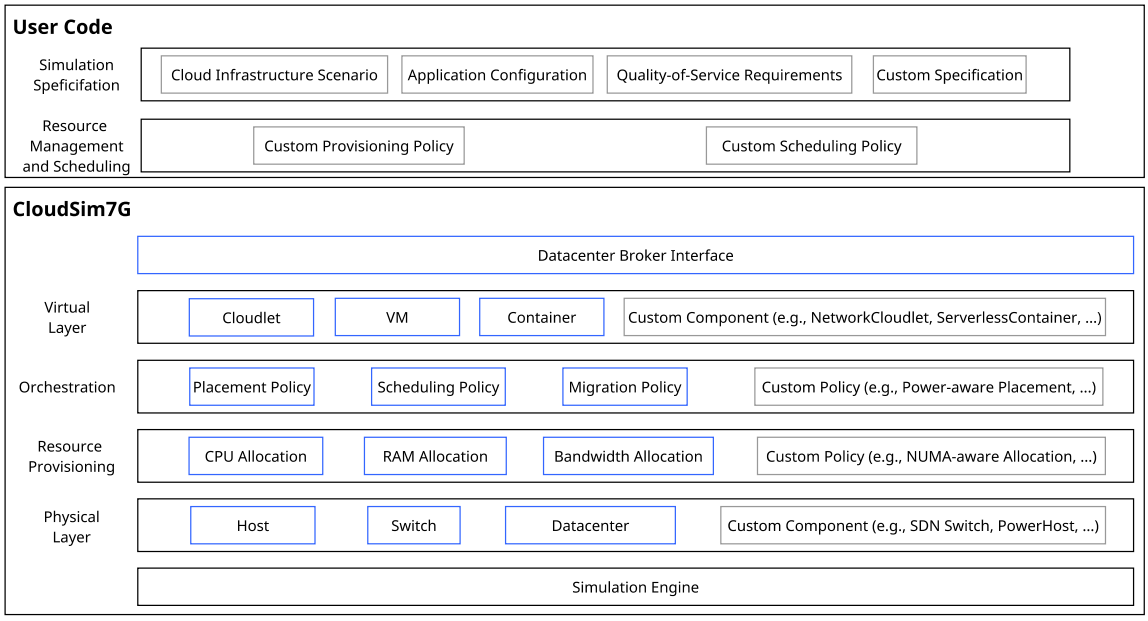


Figure 2: The architecture of CloudSim 7G. Blue boxes correspond to the CloudSim components of the base layer. Grey boxes correspond to user-defined policies and scenario configurations, as well as newly created or extended CloudSim components.

Section 4.1 gives an overview of the high-level architecture of CloudSim 7G. Section 4.2 describes in detail the new internal design of CloudSim to implement the integration layer within the building blocks of CloudSim. Section 4.3 and Section 4.4 provide a practical description of the refactoring, optimization and refinement process the codebase has undergone to accommodate the adaptation interface. Lastly, Section 4.5 gives the guidelines to update an old module to CloudSim 7G.

4.1. Architecture

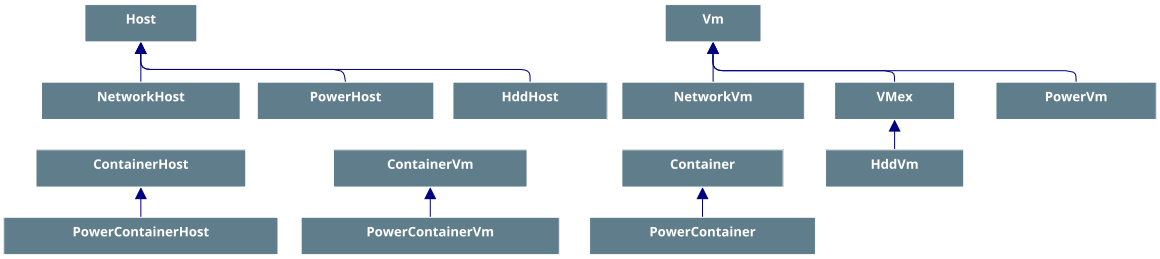
Figure 2 shows the up-to-date architectural components of CloudSim 7G. Several building blocks have been removed or consolidated compared to the figure in the seminal paper [8] thanks to the generalized interfaces of CloudSim 7G. The overall structure is the same: a user-facing section for Cloud providers, application developers, and researchers (the "User Code"), and a portion dedicated to the development of the simulator and its modules.

The user code exposes the functionalities to evaluate a workload on an infrastructure. More specifically, a user specifies: i) the characteristics of the cloud infrastructure in terms of the number of physical hosts, their hardware specification, and how they are interconnected (if the user is interested in modeling the network flow); ii) the characteristics of the virtual components and their resource requirements; iii) the characteristics of the cloud applications coupled with optional Quality-of-Service requirements; and iv) the characteristics of any installed CloudSim module. The latter are typically loaded into the user installation of CloudSim using build automation tools like Maven or Gradle. Depending on the modules in use by the CloudSim user, an application may be expressed as a workflow (using NetworkCloudSim, for instance), or a host may exhibit energy-awareness capabilities through the power-aware module. A Cloud researcher may go a step further and programmatically extends such functionalities to perform a customized and more complex evaluation of resource management techniques. These are typically more "aware" provisioning, placement, or scheduling policies: a Cloud researcher may implement its own topology-aware cloudlet scheduling policy to optimize a workflow application within a networked datacenter, or exploit the migration module to develop an auto-scaling policy for web sessions.

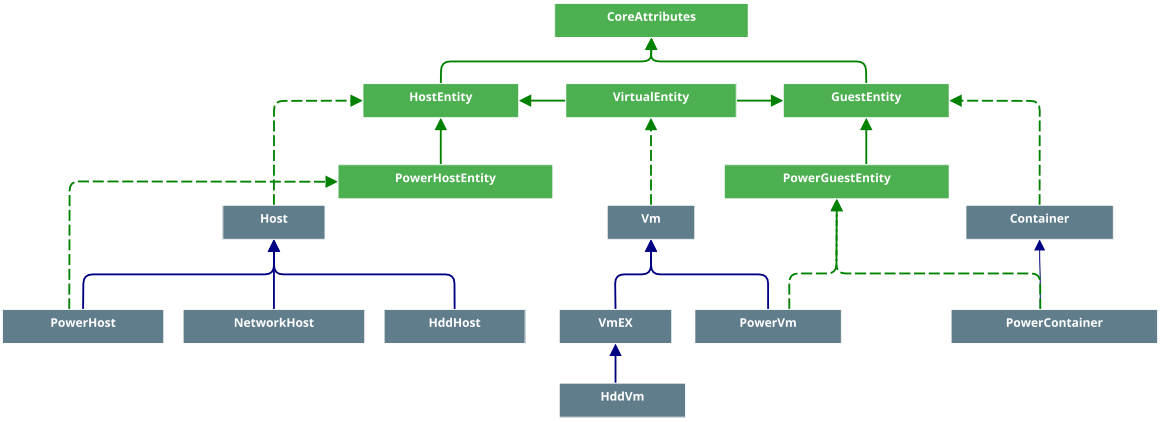
Subsequently, CloudSim configures the simulation according to the user specifications, replacing the basic building blocks with the custom policies and components expressed in user code. In particular, CloudSim 7G provides out-of-the-box support for containers, VMs, "traditional" cloudlets, and workflow applications (i.e., networked cloudlets constructed as graphs) at the virtual layer. The latter resources are then orchestrated through a collection of basic placement, scheduling, and migration policies. For instance, CloudSim's power-aware package uses migration policies to optimize the VM placement using statistical analysis of host load utilization. Throughout the whole software stack, it is possible to exploit custom components, such as the serverless capabilities of CloudSimSC, as long as they are compatible with CloudSim 7G.

4.2. Design

CloudSim is a toolkit that leverages the features of the Java programming language to offer a highly extensible and portable simulation platform. CloudSim modules are developed using inheritance and composition in Java to enhance



(a) CloudSim 6G



(b) CloudSim 7G

Figure 3: UML class diagram of host and guest entities in CloudSim. Green boxes depict Java interfaces, and blue boxes represent Java classes. The relationship arrows follow the UML relations notation style: a dashed line represents a realization/implementation, and a full line represents inheritance.

existing components or create new ones. The design changes of CloudSim 7G impacted the following Java classes (and their extensions): `Datacenter`, `DatacenterBroker`, `Host`, `Vm`, `Container`, `VmAllocationPolicy`, and `CloudletScheduler`. These classes implement the building blocks of the simulator previously described at the beginning of Section 2, and also depicted in Figure 2.

CloudSim 7G presents a collection of Java interfaces that generalizes the definition of some components and facilitates the coexistence of multiple CloudSim modules within the same simulated scenario. This design change is one of the biggest shifts in the codebase. Firstly, CloudSim 7G introduces the concept of *Guest* entity and *Host* entity. A guest entity executes actions related to the management and processing of cloudlets according to a given scheduling policy. A host entity executes actions related to the management (i.e., allocation, provisioning, and scheduling) of guest entities within the simulated cloud infrastructure. A guest entity runs inside a host entity which may be shared among several guest entities. Such sharing affects the ability of a guest entity to execute its actions. Figure 3 depicts the UML class diagram of the components affected by the design change. For instance, the authors of ContainerCloudSim had to distinguish between the `Container` and `Vm` classes. The two components performed the same actions but were implemented as completely independent entities. This triggered the need for further redundant components: an instance of the `Container` class required an extension of the `Vm` class, called `ContainerVm` (i.e., a VM that hosts containers). In turn, the latter required an extension of the `Host` class called `ContainerHost`, which can only reside on an extension of the `Datacenter` class, the `ContainerDatacenter`. To overcome such structural limitations of the framework, CloudSim 7G introduces the following Java interfaces:

1. `HostEntity`: A Java interface featuring a collection of abstract methods required to implement a host entity and its actions. A few methods provide a default, generalized implementation of an action. Therefore, a possible implementation of the `HostEntity` interface corresponds to the `Host` class in previous CloudSim versions.
2. `GuestEntity`: A Java interface featuring a collection of abstract methods required to implement a guest entity. Similarly, a few generalized implementations are provided by default. Therefore, a possible implementation of the `GuestEntity` interface corresponds to the `Vm` and `Container` classes in previous CloudSim versions.
3. `CoreAttributes`: A Java interface that defines the methods required for implementation by both `HostEntity` and `GuestEntity` classes.

4. `VirtualEntity`: A placeholder interface to implement an entity that is simultaneously a `HostEntity` and a `GuestEntity`. This interface is essential to support nested virtualization.
5. `PowerHostEntity` and `PowerGuestEntity`: Extended interfaces to standardize the integration of power-aware features into the implementation of host and guest entities.

The new design rendered some components completely redundant, such as the `ContainerHost` or `NetworkVm` classes (see Figure 3). As a result, module developers must adhere to the newly defined Java interfaces to be compatible with CloudSim 7G (More on this in Section 4.5).

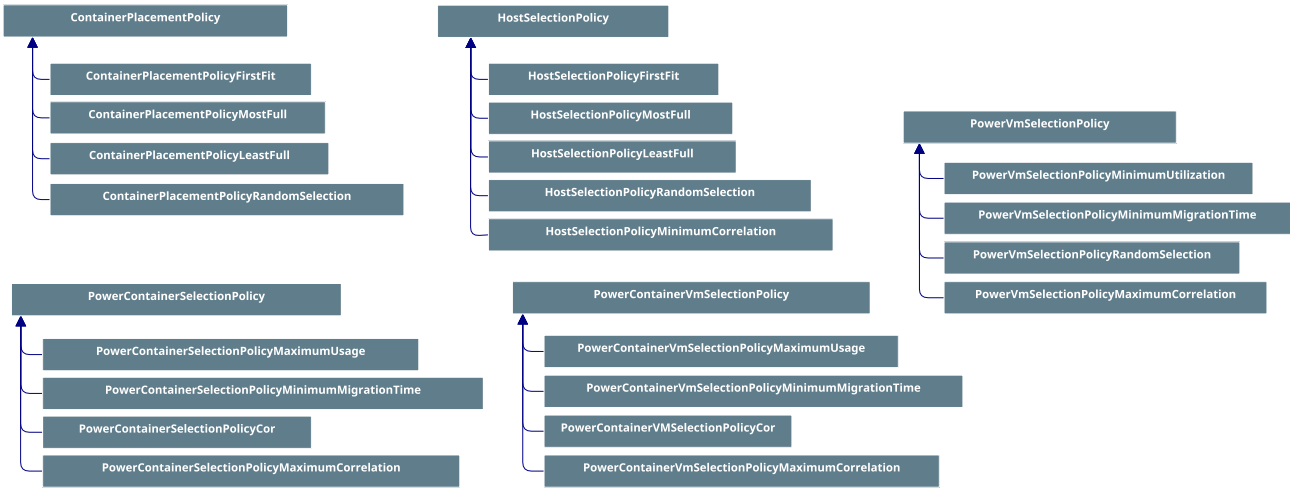
Secondly, CloudSim 7G introduces the concept of *selection policy*, which generalizes the process of selecting an entity with a criterion from within a list of candidates. This concept is typically employed to implement placement and migration policies. In previous iterations of CloudSim, there is a clear divergence between a placement policy (i.e., select a host for a VM to be placed) and a migration policy (i.e., select a VM to be migrated), despite being essentially the same activity. Therefore, the selection policy interface in CloudSim 7G serves as the fundamental building block for implementing placement, migration, or any policy that involves selecting an entity. This design change significantly simplifies the codebase, as illustrated in Figure 4, by rendering redundant several Java classes from ContainerCloudSim and the power module. Specifically, the amount of Java classes related to the selection process decreases from 26 to 11.

In conclusion, CloudSim 6G features a large number of unnecessary copied-pasted classes. As previously stated, the original version of ContainerCloudSim used to treat VMs and containers as completely independent entities, even if they performed the same actions. In comparison, CloudSim 7G provides a common collection of interfaces that unites most modules developed so far, as long as they are developed correctly using OO principles. The new internal design of CloudSim 7G helped with code deduplication: thanks to the new architecture, ContainerCloudSim has undergone a 64% reduction in terms of line of code (LoC). Similarly, the source code of NetworkCloudSim has been reduced by 50%. The power module has been reduced by 21%. The total amount of LoC removed is more than 13000, with no functionality lost during the deduplication process. This was verified by running the examples provided in the CloudSim example package.

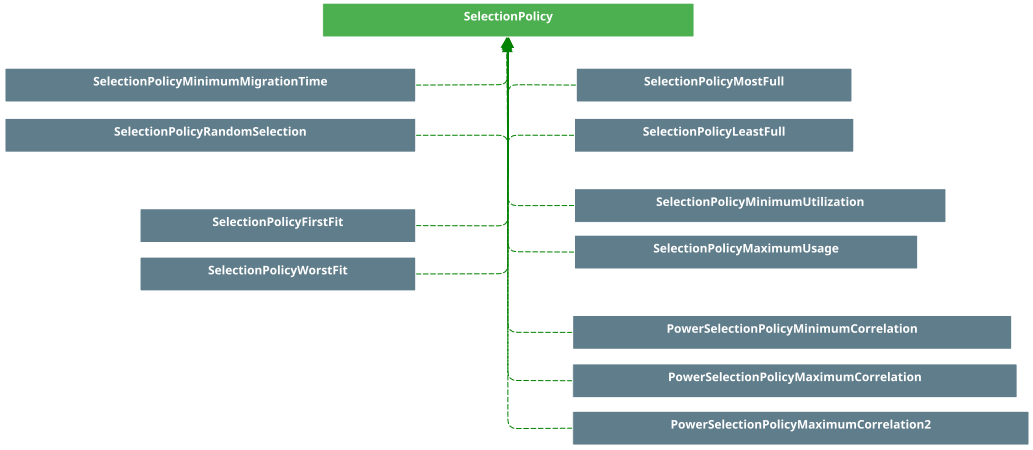
4.3. Code Refactoring and Optimization

The CloudSim refactoring process mainly focused on code deduplication thanks to the new internal design, on the improvement of code reusability, and on the enforcement of compliance with OO principles. The code optimization step comprises the following changes:

1. Removed multiple parts of the codebase that contained redundant operations or that were marked as deprecated.
2. Eliminated redundant data structures and variables in the `VmScheduler` class and its extensions, consolidating the components responsible for tracking Pe and MIPS allocations.
3. Simplified a series of nested loops that could be combined into a single loop, reducing the time complexity from $O(n^2)$ to $O(nlogn)$ in some instances.
4. Optimized the usage of built-in data structures and sorting algorithms. Depending on the use-case, choosing an appropriate data structure (and the most efficient sorting algorithm for it) significantly improves the performance of the simulator. For example, `ArrayList` and `LinkedList` have strengths and weaknesses in different operations: `Iterator.remove()` and `add()` have constant complexity in a `LinkedList` but linear complexity in an `ArrayList`; on the other hand, the `get()` operation has linear complexity in a `LinkedList` but constant one in an `ArrayList`.
5. Enforced the use of method `isEmpty()` for lists, instead of checking the "actual" size, as in certain implementations (such as `LinkedList`), determining the size requires counting the items, which is more time-consuming than simply checking for emptiness.
6. Optimized string operations by using `StringBuffer` and `StringBuilder`. During a simulation, the simulator prints many logs on the console or outputs them to log files. For convenience, many developers use the plus operator to concatenate strings. However, since Java strings are immutable, this coding behavior caused unnecessary load on the CPU and memory.
7. Preferred the use of primitive data types (e.g., `int`, `long`, `double`) rather than classes (e.g., `Integer`, `Long`, `Double`) where possible to minimize object creation and auto-boxing/auto-unboxing overheads. Furthermore, we maximized the reuse of objects to avoid the frequent creation/destruction of objects from the Java heap. This optimizes memory usage and reduces the frequency and load of garbage collection (GC).
8. Removed some use of synchronized data structures and replaced by non-synchronized counterparts, given the single-threaded nature of the core simulation engine.
9. Performed the necessary code upgrades for supporting the latest Java Development Kit (JDK), the JDK21.
10. Removed duplicated code snippets in extensions to CloudSim's core components, such as `Datacenter`, `Vm` and `DatacenterBroker`, as well as scheduling components.



(a) CloudSim 6G



(b) CloudSim 7G

Figure 4: UML class diagram of the host and guest selection policies in CloudSim. Green boxes depict Java interfaces, and blue boxes represent Java classes. The relationship arrows follow the UML relations notation style.

11. Frequently invoked methods with non-constant computational complexity were cached in local variables for efficiency. For instance, the calculation of the required per-core MIPS of VMs and containers used to be performed continuously throughout the simulation, which involved iterating through multiple arrays. Similarly, `getUid()` previously reconstructed the unique identifier from scratch with each call, involving a string concatenation.

4.4. Code Refinement

The refinement process targeted: i) the event handling system; ii) the logic of the `CloudletScheduler` class; and iii) the NetworkCloudSim module in an attempt to provide a better user experience. Detailed descriptions of each bullet point are provided in the following paragraphs, in order.

CloudSim employs a set of event tags to indicate an action that needs to be undertaken by a simulated entity when an event is triggered at run-time. Each CloudSim module typically creates its own collection of custom tags, which can potentially lead to collisions if integer values are inadvertently reused, to the detriment of multi-module scenarios. In CloudSim 7G, the event handling system has been updated to make use of the `Enum` Java class for declaring tags to improve code readability and prevent collisions.

The original version of `CloudletScheduler` abstract class was not a generic template that represented the life-cycle of a cloudlet deployed on a guest entity: any extension to the `CloudletScheduler` class needed to re-implement the whole scheduling logic, causing a lot of redundant code. As a significant side-effect, different implementations of the `Cloudlet` class could not coexist within the same cloudlet scheduler, and consequently, on the same VM (unless manually managed by the module developer, for each possible cloudlet implementation). CloudSim 7G solves this problem by providing a revised cloudlet

scheduling life-cycle represented by Algorithm 1, extracted from the `CloudletScheduler` abstract class. Lines 4, 7, and 14 expose 3 handler methods (highlighted in gray) to standardize the customization of the scheduling behaviors. The first two handlers are used to customize the update logic and stopping condition of a cloudlet. Therefore, any extension to the `Cloudlet` class is supported out-of-the-box by a `CloudletScheduler` instance. For instance, the `NetworkCloudlet` class exploits these 2 handlers to implement the stages that realize an activity within a scientific workflow. The third handler is used by the cloudlet scheduler itself to customize the unpause logic for cloudlets that are waiting to be executed. Lines 1-6 update the number of executed instructions of each active cloudlet in the time interval since the previous processing update. Lines 7-10 terminate the cloudlets that finished executing all the instructions. Line 11-13 returns early if the cloudlet scheduler has no more events (i.e., no active cloudlets) in the foreseeable future. Lines 14-16 unpause a subset of waiting cloudlets based on a customizable behavior of the cloudlet scheduler. For instance, `CloudletSchedulerTimeShared` class does not use the handler method, since all the submitted cloudlets are executed in a time-shared manner. Lines 17-23 estimate the simulation time for the next discrete event for this cloudlet scheduler, which depends on the finish time of the earliest finishing cloudlet. Thanks to this refinement process, the `CloudletScheduler` class and its extensions have undergone a 40% LoC reduction.

Algorithm 1 Revised logic of the cloudlet scheduler instantiated by the guest entities

Input: $mipsShare$, currently available processing power to the guest entity
Input: $currentTime$, current simulation time
Input: $previousTime$, simulation time of the previous scheduling update
Input: $cloudletExecList$, list of active cloudlets on the guest entity
Input: $cloudletWaitList$, list of waiting cloudlets on the guest entity
Output: Predicted completion time of the earliest finishing cloudlet, or 0 if there are no active cloudlets left

1: $timeSpan \leftarrow currentTime - previousTime$
2: **for** $cl \in cloudletExecList$ **do**
3: $allocMips \leftarrow currentlyAllocatedMipsForCloudlet(currentTime, mipsShare, cl)$
4: **if** cl.update() **then**
5: $cl.cloudletLengthSoFar \leftarrow cl.cloudletLengthSoFar + (timeSpan \cdot allocMips)$
6: **end if**
7: **if** cl.isFinished() **then**
8: $cloudletExecList.remove(cl)$
9: **end if**
10: **end for**
11: **if** $cloudletExecList.empty() \wedge cloudletWaitList.empty()$ **then**
12: **return** 0
13: **end if**
14: $unpausedCloudletList \leftarrow$ unpauseCloudlets(cloudletWaitList)
15: $cloudletWaitList.removeAll(unpausedCloudletList)$
16: $cloudletExecList.addAll(unpausedCloudletList)$
17: $nextEvent \leftarrow Double.MAX\_VALUE$
18: **for** $cl \in cloudletExecList$ **do**
19: $estFinishTime \leftarrow currentTime + \frac{cl.cloudletLength - cl.cloudletLengthSoFar}{currentlyAllocatedMipsForCloudlet(currentTime, mipsShare, cl)}$
20: **if** $estFinishTime < nextEvent$ **then**
21: $nextEvent \leftarrow estFinishTime$
22: **end if**
23: **end for**
24: **return** $nextEvent$

Regarding NetworkCloudSim, it introduced a series of network-related features that were cumbersome to set up, and some features were not fully implemented, leading to confusion and difficulty in their use. Moreover, there were plenty of hard-coded constants that could not be customized. In short, NetworkCloudSim is presented as a proof of concept, rather than a usable module. To this end, CloudSim 7G includes a revised version of NetworkCloudSim, which has been generalized to function as a proper module, and enriched with several new features to complete and enhance the user experience.

For instance, there was no user-friendly way to configure a network with multiple switches. As a result, a CloudSim user had to directly access the member variables of a `Switch` instance, which was very error-prone. Furthermore, the `NetworkCloudlet` class, which implemented the cloudlets with network capabilities for constructing workflow applications, followed a different execution model than the traditional cloudlet, despite inheriting the properties from the `Cloudlet` class. A stage within a networked cloudlet was defined in terms of execution time (in milliseconds), whereas a traditional cloudlet is defined in terms of execution length (in millions of instructions). The payload size of a packet was defined in bytes, but it

Algorithm	Total Heap Usage (MB)			Run-time (s)		
	6G	7G	**Reduction**	6G	7G	**Reduction**
Dvfs	2608.629	2420.015	7.2%	11.071	10.677	3.6%
MadMmt	43531.363	44182.750	-1.5%	104.666	102.666	2%
ThrMu	43400.920	35601.881	18%	99	96	3%
IqrRr	41549.674	33458.236	19.5%	101.333	93.333	8%
LrrMc	41452.075	36243.197	12.6%	92.333	88.333	4%

Table 2
Performance comparison of CloudSim 6G and CloudSim 7G in terms of total allocated memory and run-time. Each experiment is performed 3 times, and the average result is shown in the table. The "Reduction" column displays the percentage decrease (or increase, if the value is positive) for each metric.

was not converted to bits when calculating the packet transmission time. NetworkCloudSim allowed configuring a deadline to the execution time of the workflow application, but it did not implement the logic to check whether the deadline had been met.

4.5. Guidelines for Upgrading a Module to CloudSim 7G

This section provides the guidelines for restoring the compatibility of older CloudSim modules with the new base layer of CloudSim 7G. Most re-engineered Java methods within classes for core CloudSim components have retained their original APIs, some kept the previous one as a deprecated alternative. While we aimed to minimize disruptions, the new internal design of CloudSim 7G necessitated key changes that need to be addressed by extension developers; Using modern IDEs should make the upgrade process piloted and straight-forward, as they can detect dangling code paths and the use of deprecated methods.

Firstly, some classes have been removed or replaced: for instance, the `ResCloudlet` class has been completely integrated into the `Cloudlet` class. Consequently, any extension to the `ResCloudlet` class now needs to inherit its properties from the `Cloudlet` class. Secondly, some methods using Java generics require conversion from type `Host` to `HostEntity` (or from `Vm` to `GuestEntity`) due to ambiguities caused by the type erasure mechanism.

Extensions to allocation and migration policies must adhere to the novel API offered by the selection policy interface. Lastly, since the tag system in CloudSim has been entirely rewritten to use the `Enum` class instead of integer primitives, extension developers will need to update their custom tags accordingly. As a side note: although the novel template-based `CloudletScheduler` class maintains full backward compatibility, we encourage module developers to adhere to the new framework outlined in Algorithm 1, thus integrating their custom changes to the scheduling life-cycle through the proposed handler methods.

The removal of the redundant data structures for the allocation of hardware resources for VMs within the `VmScheduler` class and its extensions (described in Section 4.3), required some variable renaming to offer a more generalized API. The `VmScheduler` class exposes the following structure: `peAllocationMap` and `mipsAllocationMap`, instead of `peMap` and `mipsMap`. More specifically, Cloudhence why we `peAllocationMap`in `VmAllocationSpaceShared`, since it was a redundant copy on `peMap`

5. Performance Evaluation

In CloudSim 7G, the codebase has undergone massive refactoring to improve the readability and performance of the code. This section presents a series of experiments conducted on an x86-64 laptop with Intel i7-8665U CPU, 8MB L3 cache, 1MB L2 cache, 16GB RAM, and Ubuntu 24.04 LTS. The goal is to evaluate the performance improvements of CloudSim 7G, compared to CloudSim 6G, using large trace datasets, despite the introduction of new abstraction layers. We utilized OpenJDK21 in both CloudSim 6G and CloudSim 7G, namely OpenJDK 64-Bit Server VM build 21.0.4+7-Ubuntu-1ubuntu224.04. All experiments are conducted with JVM parameters `-Xms4G -Xmx4G` (i.e., fixed heap size to avoid dynamic resizing at run-time) using the default garbage-first garbage collector [14].

We collected the total memory allocated by each experiment at run-time by aggregating the reclaimed heap memory reported in Java Garbage Collection log messages (i.e., JVM parameter `-Xlog:gc*`). The wall-clock run-time is obtained from the execution summary provided by Maven, the build automation tool used by CloudSim. All experiments are conducted with CPU frequency blocked at 1.90 GHz (the base frequency suggested by the vendor), and no turbo or dynamic boosting. For each algorithm and metric combination, we conducted the experiment 3 times and computed the average heap usage and wall-clock run-time.

For the sake of reproducibility, we evaluate the performance improvements of CloudSim 7G using a collection of simple heuristic algorithms from the power module[5] and workload traces sampled from the PlanetLab package [35], both of which are readily available within the CloudSim package. Each experiment involves a distinct VM allocation and selection policy for the study of energy and performance-efficient dynamic consolidation of VMs within a cloud datacenter.

(a) Simulated datacenter.

(b) Simulated workflow application with end-to-end deadline.

Figure 5: Case-study: A simple workflow application deployed on a datacenter of interconnected physical hosts.

Table 2 depicts the results of the performance comparison: CloudSim 7G consistently outperforms CloudSim 6G in terms of average total heap usage and wall-clock run-time, except for MadMmt (although the difference is negligible given the run-time improvement). In selected experiments, CloudSim 7G reduces run-time by up to 8 seconds and lowers total memory allocated by as much as 8091 MB, greatly improving the performance of the simulator.

6. Case Study

This section presents a simulation scenario that uses multiple CloudSim modules. In particular, the following entities will interact in a multi-module scenario: i) the "traditional" VMs provided by CloudSim since the first version; ii) the network and application model of the overhauled NetworkCloudSim, and iii) the service broker of CloudSimEX for simulating realistic cloudlet arrivals. Notice that the goal of this section is not to thoroughly evaluate a set of placement or scheduling strategies to optimize a deployment, but to demonstrate CloudSim 7G's new multi-module feature with a simple scenario.

In the following, we take the viewpoint of a private Cloud provider that needs to deploy a workflow application of interconnected tasks, represented as a Directed Acyclic Graph (DAG). The provider's goal is to estimate the end-to-end (E2E) deadline for its application based on some known parameters: the number of instructions per task, the payload to be transferred between tasks, and the inter-arrival time of new requests. This use-case is notoriously faced by telco providers in the context of Industry 4.0, where time-critical applications characterized by strict timing and reliability constraints are being deployed on contemporary cloud infrastructures. The multi-tenant nature of traditional cloud infrastructures poses a challenge for such an emerging use-case due to the "noisy neighbor" problem [1, 18]. Notice that previous versions of CloudSim do not support such a peculiar scenario. A series of experiments are performed to analyze how the total response time of the DAG, which is the time taken for the execution of all the tasks (the so-called makespan), is affected by the placement strategy, the network delay, and the computational noise of co-located cloudlets.

The private datacenter in Figure 5a comprises 4 homogeneous physical hosts interconnected via 2 switches. The network topology resembles a basic tree-like structure: the hosts are split equally between 2 different racks, and each group of hosts is connected to a Top-of-Rack (ToR) switch using a symmetrical gigabit connection. The ToR switches communicate through an Aggregate Switch via a symmetrical gigabit connection as well. The logical representation of the workflow application is depicted in Figure 5b: a DAG with two tasks, $T0$ and $T1$, the source and sink node of the graph, respectively, connected in a chain by a data transfer dependency. The DAG is "activated" by user requests that trigger periodically. These requests are simulated in CloudSim by submitting two networked cloudlets interconnected according to the given topology. Hence a "DAG activation" is a periodic event that refers to the following chain of activities: i) $T0$ and $T1$ are submitted for execution to the datacenter; ii) $T0$ executes for a pre-fixed amount of millions of instructions L_{T0}; iii) $T0$ transmits a payload to $T1$ of fixed-size $payloadSize$; and iv) $T1$ executes for a pre-fixed amount of L_{T1}.

For the sake of simplicity, we will only focus on the effect of cloudlet scheduling and placement using two payload sizes for data transmission. Therefore we assume hosts with infinite capabilities that meet the VMs' requirements in terms of CPU, memory, and network bandwidth. Additionally, the VMs are big enough to accommodate all the cloudlets representing the workflow application together, if necessary. We employ a time-shared cloudlet scheduling to simulate computational noise for co-located cloudlets, and the inter-arrival time of the cloudlets is sampled from an exponential distribution and distributed across selected VMs in a round-robin fashion. The experimental evaluation focuses on 3 possible cloudlet placement configurations: I) $T0$ and $T1$ are co-located, meaning that data is transmitted locally without using the physical network; II) $T0$ and $T1$ are located on the same rack, meaning that data is routed through a ToR switch only; and III) $T0$ and $T1$ are located on different racks so that data must be transmitted through the aggregate switch as well.

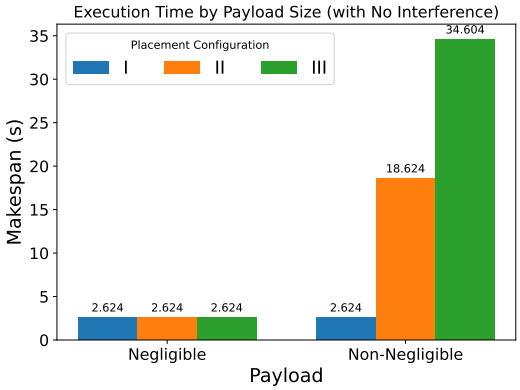


Figure 6: Makespan of a single activation of the workflow application in function of the payload size.

The VMs are modeled after the new AWS "m7g.medium" general-purpose EC2 instances[2], which are single-core VMs powered by the AWS Graviton3 processor where the virtual CPU is a physical core running at 2.6 GHz. The processing power can be approximately estimated in MIPS using the following formula, which is derived from the textbook definition of CPU time [36]:

$$vmMips = \frac{clk\_rate \cdot IPC}{10^6} \tag{1}$$

where clk_rate is the CPU frequency in Hz, and IPC is the number of instructions per cycle. In our use-case, the processing power can be roughly estimated to $vmMips = 7800$ MIPS, assuming $IPC = 3$. The execution lengths of the cloudlets are configured as $L_{T0} = L_{T1} = 10000$ MI to ensure a sufficiently long completion time for presentation purposes. The packets exchanged between nodes include either a negligible payload of 1 byte, or a non-negligible payload of 1 gigabyte of data (i.e., $payloadSize \in \{1B, 1GB\}$).

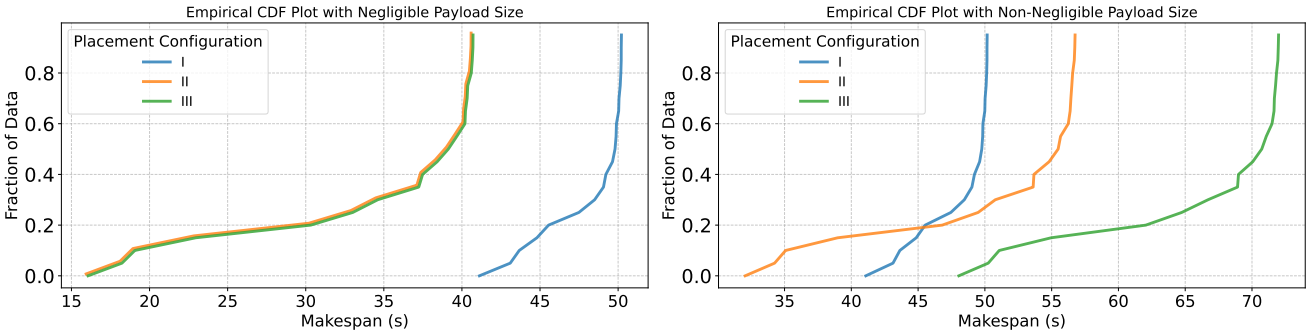
The deployed workflow application must meet a desired E2E latency specified as a deadline by the Cloud provider. The tightest possible deadline equals the makespan of a DAG activation, which takes into account the compute times, as well as the transmission times due to the network switching delay. Other factors can degrade the processing power and the average network bandwidth of VMs and switches, such as: i) the interference of co-located cloudlets; ii) components with heterogeneous hardware capabilities; and iii) more complex network and DAG topologies. A detailed analysis is necessary in such cases to determine the worst-case execution time for each cloudlet and appropriately "inflate" the expected makespan of the activation [2]. However, this is out-of-scope, hence why for such a simple scenario the makespan can be estimated as:

$$makespan = \left(\frac{L_{T0}}{vmMips} + \frac{L_{T1}}{vmMips}\right) + networkHops \cdot \left(\frac{payloadSize}{sndBw} + \frac{payloadSize}{rcvBw}\right), \tag{2}$$

where $sndBw$ is the network bandwidth for sending packets and $recvBw$ is the network bandwidth for receiving packets; Both are equivalent to 1 gigabit given the components of our simulated datacenter (i.e., VMs and switches). The number of network hops required to realize the communication is $networkHops$, and it depends on the cloudlet placement: 0 hops for Configuration I, 1 for Configuration II, and 2 for Configuration III. Equation (2) should be enough to assess the correctness of the network module.

Figure 6 illustrates some preliminary experiments where only a single DAG activation is simulated. For a negligible payload size (i.e., $payloadSize = 1B$), there is no difference between each placement configuration, as the network delay is irrelevant. For a non-negligible payload size (i.e., $payloadSize = 1GB$), each network hop adds a constant delay of ~16 seconds. The simulated network delay aligns with the estimation from Equation (2) when applied to our use-case parameters, resulting in $makespan = 2.564 + networkHops \cdot 16$. Figure 7 depicts the empirical Cumulative Distribution Function (CDF) of the makespans for a series of DAG activations. In particular, the DAG is activated 20 times with exponential inter-arrival times so that the computational noise increases incrementally due to the number of concurrent activations. For a negligible payload size, Configuration I exhibits significant computational noise as depicted in Figure 7a, with a median value that is 25% higher than those of Configuration II and Configuration III. Notice that there is no difference between Configuration II and III due to the negligible payload; For presentation purposes only, the orange curve has been slightly shifted upwards to remove the overlap. This occurs because all cloudlets reside on the same VM: the exponential inter-arrival time of user requests triggers increasingly overlapping activations of both the source and sink nodes of the DAG, leading to high contention. Consequently,

[2]See: https://aws.amazon.com/ec2/instance-types/m7g/.



(a) Negligible Payload Size.

(b) Non-Negligible Payload Size.

Figure 7: Multiple activations of the workflow application in function of the payload size.

the VM becomes progressively congested due to the shared processing power. On the other hand, distributing the workload across two separate VMs significantly reduces the pressure, resulting in less contention and therefore shorter makespans, even with the exponential inter-arrival time. Figure 7b shows the result for a non-negligible payload size. Notice that Configuration I exhibits the same behavior as before since the network is not used. Configuration II and III exhibit a similar behavior too, but the curves are shifted to the right and separated from each other, showing higher makespans in Configuration III due to the additional network hop. This experiment gives more leverage to the Cloud provider in estimating an end-to-end deadline to minimize the number of missed deadlines. For instance, a deadline of 70 seconds would likely guarantee no misses, though it might be too lax depending on the specific context. A deadline of 50 seconds is optimal only with Configuration I, however, it presents shortcomings in terms of fault-tolerance.

7. Conclusions and Future Outlook

This article unveils a revival of the CloudSim project with the release of its seventh iteration. CloudSim 7G is the release with the most internal changes to date: it offers a novel base layer comprising multiple past modules that have undergone a massive refactoring, optimization and refinement process to accommodate the proposed design change. CloudSim 7G facilitates the integration of multiple modules in the same simulated scenario, opening new opportunities for the evaluation of scheduling and resource management techniques for next-generation Cloud Computing environments. Furthermore, we have empirically demonstrated the improved performance of CloudSim 7G thanks to the aforementioned refactoring and refinement process. Thanks to our recent optimization effort, CloudSim 7G uses 20% less total memory allocations in selected experiments and overall saves 4-10 seconds compared to CloudSim 6G.

In what follows, we describe a series of future courses of action that we hope will improve CloudSim with the help of the research community. Several researchers [47, 43, 38] have criticized the cloudlet scheduling component accuracy for evaluating production-quality Cloud components. In this regard, CloudSim 7G represents a significant advancement for accelerating the development of new extensions, including one offering a comprehensive overhaul of the scheduling component so to replace the overly simplistic default models.

A further development step of CloudSim's internals concerns the merging of the host and guest entity components into one standardized interface. Indeed, the two concepts realize the same activities, but at different layers: a host entity resides on the "physical" layer, whereas a guest entity resides on the "virtual" layer. Such a paradigm shift is currently too catastrophic for restoring compatibility with older modules. In this regard, CloudSim 7G serves as an intermediate step toward the realization of a fully generic physical/virtual entity component in future releases. Secondly, CloudSim might be updated to specify the computational capabilities of hosts and virtual machines in terms of clock frequency, in addition to the current MIPS performance metric. Lastly, CloudSim requires a more comprehensive test suite to ensure its correctness and prevent software regressions between releases.

Given the large research work on approaches based on Machine Learning (ML), Artificial Intelligence (AI) and Reinforcement Learning (RL) [25], it may be useful to explore the possibility of integrating within CloudSim additional components to train and evaluate AI/ML models, to simulate AI-enhanced resource management strategies in Cloud Computing. This will require the realization of new interfaces to interact with ML/AI accelerators and development frameworks.

To conclude, with the rapid growth of Cloud Computing, we encourage the research community to develop new modules that take advantage of the design shift introduced with CloudSim 7G and to investigate novel scheduling policies with

hybrid scenarios using network modeling, power, serverless, NUMA architectures, quantum computing, and VM-container-host placement policies. On a similar note, we kindly request experienced module developers to update their contributions following the guidelines provided in this paper to leverage the novel features of CloudSim 7G.

Software availability
The source code and examples of the CloudSim7G toolkit are accessible on GitHub[3] as an open-source tool under the GPL-3.0 license.

Acknowledgments
The authors acknowledge and thank all contributors and developers of CloudSim, and their mpodules that have been integrated into this new version of CloudSim. This work is partially supported by the Australian Research Council Discovery Project and EU Erasmus+ Traineeship.

Conflict of interest statement
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] L. Abeni, R. Andreoli, H. Gustafsson, R. Mini, and T. Cucinotta. Fault tolerance in real-time cloud computing. In *Proceedings of the 2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 170–175, 2023.

[2] R. Andreoli, H. Gustafsson, L. Abeni, R. Mini, and T. Cucinotta. Optimal deployment of cloud-native applications with fault-tolerance and time-critical end-to-end constraints. In *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*, UCC '23, New York, NY, USA, 2024. Association for Computing Machinery.

[3] A. Awan, M. Aleem, A. Hussain, and R. Prodan. Dfarm: a deadline-aware fault-tolerant scheduler for cloud computing. *Cluster Computing*, pages 1–22, 2024.

[4] I. Bambrik. A survey on cloud computing simulation and modeling. *SN Comput. Sci.*, 1(5), aug 2020.

[5] A. Beloglazov and R. Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[6] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1175–1220, 2002.

[7] R. Buyya, R. Ranjan, and R. N. Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *Proceedings of the 2009 international conference on high performance computing & simulation*, pages 1–11. IEEE, 2009.

[8] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya. 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, 2011.

[9] P. C. Cañizares, A. Núñez, A. Bernal, M. E. Cambronero, and A. Barker. Simcan2cloud: a discrete-event-based simulator for modelling and simulating cloud computing infrastructures. *Journal of Cloud Computing*, 12(1):133, 2023.

[10] W. Chen and E. Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *Proceedings of the 2012 IEEE 8th international conference on E-science*, pages 1–8. IEEE, 2012.

[11] T. Cucinotta and A. Santogidis. CloudNetSim – Simulation of Real-Time Cloud Computing Applications. In *Proceedings of the 4th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2013)*, 2013.

[12] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[13] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[14] D. Detlefs, C. Flood, S. Heller, and T. Printezis. Garbage-first garbage collection. In *Proceedings of the 4th international symposium on Memory management*, pages 37–48, 2004.

[15] M. Ersue. Etsi nfv management and orchestration-an overview. *Presentation at the IETF*, 88, 2013.

[16] F. Fakhfakh, H. H. Kacem, and A. H. Kacem. Cloudsim4dwf: A cloudsim-extension for simulating dynamic workflows in a cloud environment. In *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 195–202. IEEE, 2017.

[17] D. Gannon, R. Barga, and N. Sundaresan. Cloud-native applications. *IEEE Cloud Computing*, 4(5):16–21, 2017.

[18] M. García-Valls, T. Cucinotta, and C. Lu. Challenges in real-time virtualization and predictable cloud computing. *Journal of Systems Architecture*, 60(9):726–740, 2014.

[19] S. K. Garg and R. Buyya. Networkcloudsim: Modelling parallel applications in cloud simulations. In *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 105–113, 2011.

[20] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

[21] M. Jammal, H. Hawilo, A. Kanso, and A. Shami. Ace: Availability-aware cloudsim extension. *IEEE Transactions on Network and Service Management*, 15(4):1586–1599, 2018.

[22] D. N. Jha, K. Alwasel, A. Alshoshan, X. Huang, R. K. Naha, S. K. Battula, S. Garg, D. Puthal, P. James, A. Zomaya, et al. Iotsim-edge: a simulation framework for modeling the behavior of internet of things and edge computing environments. *Software: Practice and Experience*, 50(6):844–867, 2020.

[23] D. Kliazovich, P. Bouvry, and S. U. Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62:1263–1283, 2012.

[3] See: https://github.com/Cloudslab/cloudsim/releases/tag/7.0

[24] P. Kumari and P. Kaur. A survey of fault tolerance in cloud computing. *Journal of King Saud University-Computer and Information Sciences*, 33(10):1159–1176, 2021.

[25] G. Lanciano, R. Andreoli, T. Cucinotta, D. Bacciu, and A. Passarella. A 2-phase strategy for intelligent cloud operations. *IEEE Access*, pages 1–1, 2023.

[26] Y. Liu, J. K. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi. *Data center networks: Topologies, architectures and fault-tolerance characteristics*. Springer Science & Business Media, 2013.

[27] B. Louis, K. Mitra, S. Saguna, and C. Åhlund. Cloudsimdisk: Energy-aware storage simulation in cloudsim. In *Proceedings of the 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, pages 11–15. IEEE, 2015.

[28] R. Mahmud, S. Pallewatta, M. Goudarzi, and R. Buyya. Ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice management in edge and fog computing environments. *Journal of Systems and Software*, 190:111351, 2022.

[29] A. W. Malik, K. Bilal, K. Aziz, D. Kliazovich, N. Ghani, S. U. Khan, and R. Buyya. Cloudnetsim++: A toolkit for data center simulations in omnet++. In *Proceedings of the 2014 11th Annual High Capacity Optical Networks and Emerging/Enabling Technologies (Photonics for Energy)*, pages 104–108, 2014.

[30] A. Mampage and R. Buyya. Cloudsimsc: A toolkit for modeling and simulation of serverless computing environments. In *Proceedings of the 2023 IEEE 25th International Conferences on High Performance Computing and Communications (HPCC)*. IEEE, 2023.

[31] N. Mansouri, R. Ghafari, and B. M. H. Zade. Cloud computing simulators: A comprehensive review. *Simulation Modelling Practice and Theory*, 104:102144, 2020.

[32] A. Markus and A. Kertesz. A survey and taxonomy of simulation environments modelling fog computing. *Simulation Modelling Practice and Theory*, 101:102042, 2020. Modeling and Simulation of Fog Computing.

[33] H. T. Nguyen, M. Usman, and R. Buyya. iquantum: A toolkit for modeling and simulation of quantum computing environments. *Software: Practice and Experience*, 54(6):1141–1171, 2024.

[34] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi. Cloud container technologies: A state-of-the-art review. *IEEE Transactions on Cloud Computing*, 7(3):677–692, 2019.

[35] K. Park and V. S. Pai. Comon: a mostly-scalable monitoring system for planetlab. *ACM SIGOPS operating systems review*, 40(1):65–74, 2006.

[36] D. A. Patterson and J. L. Hennessy. *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann, 2016.

[37] S. F. Piraghaj, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya. Containercloudsim: An environment for modeling and simulation of containers in cloud data centers. *Software: Practice and Experience*, 47(4):505–521, 2017.

[38] A. Pucher, E. Gul, R. Wolski, and C. Krintz. Using trustworthy simulation to engineer cloud schedulers. In *2015 IEEE International Conference on Cloud Engineering*, pages 256–265, 2015.

[39] P. Raith, T. Rausch, A. Furutanpey, and S. Dustdar. faas-sim: A trace-driven simulation framework for serverless edge computing platforms. *Software: Practice and Experience*, 53(12):2327–2361, 2023.

[40] N. Saleh and M. Mashaly. A dynamic simulation environment for container-based cloud data centers using containercloudsim. In *Proceedings of the 2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 332–336, 2019.

[41] A. Satpathy, M. N. Sahoo, C. Swain, M. Bilal, S. Bakshi, and H. Song. Gamap: A genetic algorithm-based effective virtual data center re-embedding strategy. *IEEE Transactions on Green Communications and Networking*, pages 1–1, 2023.

[42] Y. Semenov and O. Sukhoroslov. Dslab faas: Fast and accurate simulation of faas clouds. *Physics of Particles and Nuclei*, 55(3):485–488, 2024.

[43] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire. Cloudsim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. In *Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 400–406, 2017.

[44] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya. Cloudsimsdn: Modeling and simulation of software-defined cloud data centers. In *Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 475–484. IEEE, 2015.

[45] J. Son, T. He, and R. Buyya. Cloudsimsdn-nfv: Modeling and simulation of network function virtualization and service function chaining in edge computing environments. *Software: Practice and Experience*, 49(12):1748–1764, 2019.

[46] C. Sonmez, A. Ozgovde, and C. Ersoy. Edgecloudsim: An environment for performance evaluation of edge computing systems. *Transactions on Emerging Telecommunications Technologies*, 29(11):e3493, 2018.

[47] O. Sukhoroslov and A. Vetrov. Towards fast and flexible simulation of cloud resource management. In *2022 International Conference on Modern Network Technologies (MoNeTec)*, pages 1–8, Oct 2022.

[48] J. Sun, T. Wo, X. Liu, R. Cheng, X. Mou, X. Guo, H. Cai, and R. Buyya. Cloudsimsfc: Simulating service function chains in multi-domain service networks. *Simulation Modelling Practice and Theory*, 120:102597, 2022.

[49] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE transactions on parallel and distributed systems*, 13(3):260–274, 2002.

[50] A. Varga and R. Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 2010.

[51] L. Wu, S. K. Garg, and R. Buyya. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 195–204, 2011.

[52] M. Ye, A. Ruocco, D. Buono, J. Bottomley, and H. Franke. Free the turtles: Removing nested virtualization for performance and confidentiality in the cloud. In *Proceedings of the 2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pages 275–281. IEEE, 2023.

[53] X. Zeng, S. K. Garg, P. Strazdins, P. P. Jayaraman, D. Georgakopoulos, and R. Ranjan. Iotsim: A simulator for analysing iot applications. *Journal of Systems Architecture*, 72:93–107, 2017.

[54] A. Zhou, S. Wang, C. Yang, L. Sun, Q. Sun, and F. Yang. Ftcloudsim: support for cloud service reliability enhancement simulation. *International Journal of Web and Grid Services*, 11(4):347–361, 2015.