

# CloudSim express: A novel framework for rapid low code simulation of cloud computing environments

Tharindu B. Hewage<sup>1</sup>  | Shashikant Ilager<sup>2</sup> | Maria A. Rodriguez<sup>1</sup> | Rajkumar Buyya<sup>1</sup> 

<sup>1</sup>The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, Victoria, Australia

<sup>2</sup>Vienna University of Technology (TU Wien), Vienna, Austria

## Correspondence

Tharindu B. Hewage, The Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Melbourne, VIC, Australia.

Email:

[tsaryakarahe@student.unimelb.edu.au](mailto:tsaryakarahe@student.unimelb.edu.au)

## Abstract

Cloud computing environment simulators enable cost-effective experimentation of novel infrastructure designs and management approaches by avoiding significant costs incurred from repetitive deployments in real Cloud platforms. However, widely used Cloud environment simulators compromise on usability due to complexities in design and configuration, along with the added overhead of programming language expertise. Existing approaches attempting to reduce this overhead, such as script-based simulators and graphical user interface (GUI) based simulators, often compromise on the extensibility of the simulator. Simulator extensibility allows for customization at a fine-grained level, thus reducing it significantly affects flexibility in creating simulations. To address these challenges, we propose an architectural framework to enable human-readable script-based simulations in existing Cloud environment simulators while minimizing the impact on simulator extensibility. We implement the proposed framework for the widely used Cloud environment simulator, the CloudSim toolkit, and compare it against state-of-the-art baselines using a practical use case. The resulting framework, called *CloudSim Express*, achieves extensible simulations while surpassing baselines with over a 71.43% reduction in code complexity and an 89.42% reduction in lines of code.

## KEYWORDS

cloud Computing, CloudSim, modelling and simulation, programming productivity

## 1 | INTRODUCTION

Cloud computing evolves at a fast rate, facilitating a wide variety of applications and services.<sup>1</sup> This leads to the continuing development of novel infrastructure designs and management approaches. However, validating such approaches in actual Cloud environments incurs significant costs, time, and effort due to repetitive testing. Consequently, Cloud environment simulators are used to explore, analyze, and stress such approaches before deploying them in actual environments.

**Abbreviations:** API, Application Programming Interface; CSV, Comma-separated Values; GITS, Generic Input Template for Cloud Simulators; GUI, Graphical User Interface; JAR, Java Archive; JSON, JavaScript Object Notation; PoJo, Plain Old Java Object; UML, Unified Modeling Language; VM, Virtual Machine; YAML, YAML ain't markup language.

This is an open access article under the terms of the [Creative Commons Attribution](https://creativecommons.org/licenses/by/4.0/) License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

© 2023 The Authors. *Software: Practice and Experience* published by John Wiley & Sons Ltd.

Simulations allow for the validation of strategies through repetitive experiments with fewer resources.<sup>2</sup> For example, novel resource allocation strategies in hyper-scale Cloud environments need to be iteratively benchmarked. Performing these tests in a real-world setting would result in significant financial costs, development time, and effort compared to less resource-intensive simulators.<sup>2</sup> Established Cloud environment simulators are generally preferred for simulated scenarios, primarily due to their feature availability.<sup>3</sup> However, existing simulators are bound to a specific programming language and its ecosystem. This requires designing simulation scenarios via code, performing various configurations, and recompiling the code upon changing the simulation.<sup>4,6</sup> Furthermore, a simulation involves maintaining a significant amount of code, making it complex to find defects unless appropriate programming practices are followed.<sup>7</sup>

On the other hand, script-based approaches are preferred due to their simplicity. For example, in the field of machine learning research, script-based development tools such as Jupyter Notebooks are widely used.<sup>8</sup> Similarly, multiple approaches have been proposed to reduce the overhead of programming language expertise in Cloud environment simulation using script-based<sup>9-11</sup> or graphical user interface (GUI)-based<sup>3,10,12-15</sup> Cloud environment modeling. In doing so, both script-based and GUI-based approaches compromise on the extensibility of the simulation platform. For example, Silva et al.<sup>11</sup> propose a human-readable script to define the Cloud environment, which is then consumed by a specific software tool to implement the simulation scenario. However, the script is limited to a specific set of allocation policies that can be used. Similarly, DartCSim<sup>3</sup> attempts to customize the simulation platform by providing an embedded code editor, but only a limited set of methods are allowed to be overridden. Compromised extensibility reduces the flexibility in creating simulation scenarios. This motivates us to investigate reducing programming language overhead in simulating Cloud computing environments, with a minimum compromise in simulation platform extensibility.

In this article, we propose an architectural framework to translate a script-based Cloud computing environment into an actual simulation in a simulation platform. Our framework segregates the simulation platform and translation logic into different layers, preserving the extensibility of the simulation platform while providing a simplified script-based simulation. To validate the proposed framework, we implement it for the widely used Cloud environment simulator, the CloudSim toolkit.<sup>4</sup> The resulting framework, called *CloudSim Express*, enables human-readable script-based Cloud environment simulations. It consumes a YAML (YAML Ain't Markup Language)<sup>16</sup> script that describes the simulation system model in a top-down approach and automates code-based simulation scenario implementation. It separates the simulation platform extension logic by allowing dynamic injection of extensible modules via the YAML script. Furthermore, it utilizes the layered architecture of the proposed framework to configure extensible modules with the CloudSim toolkit.

We evaluate the performance of *CloudSim Express* against the state-of-the-art baselines using a practical use case of Cloud environment simulation. Unlike the baselines, our *CloudSim Express* framework makes minimal compromise in simulation platform extensibility while significantly reducing the overhead of programming language expertise. *CloudSim Express* outperforms the baselines with a reduction of over 71.43% in code complexity and over 89.42% in lines of code. Additionally, for the purpose of reproducibility and broader use by the research community, the developed solution is released as an open-source project. In summary, our contributions are,

- We propose an architectural framework to translate an script-based Cloud environment system model, into a concrete simulation
- We validate the proposed architectural framework by implementing it for the widely-adopted CloudSim toolkit<sup>4</sup>
- We release the implemented framework, the *CloudSim Express*, as an open source framework

The rest of the article is organized as follows. Section 2 discusses related background, Section 3 discusses our proposed Architectural framework, Section 4 discusses our implementation of the proposed framework, Section 5 discusses the comparison and the evaluation of the implementation, Section 6 discusses the related literature, and Section 7 concludes our work with potential future work.

## 2 | BACKGROUND

### 2.1 | Cloud environment and simulation

A Cloud environment is a large and complex system composed of users, computing resources, network facilities, and applications. It focuses on various characteristics, including geographic location awareness, low latency, application

scalability, and elasticity. As it evolves, new challenges emerge, such as energy efficiency and load balancing.<sup>2</sup> However, conducting experiments in actual Cloud environments is expensive, time-consuming, and difficult to replicate. Furthermore, developers lack control over Cloud environments, making it infeasible to repeat benchmark scenarios.<sup>2</sup>

Simulation allows for the imitation of real-world Cloud environments and enables experimentation with their complex internal interactions.<sup>17</sup> It enables the stress-testing and iterative improvement of new strategies through repeated benchmark scenarios. Moreover, simulation requires fewer resources and less time to conduct experiments. Therefore, the use of Cloud environment simulators for experimenting with new strategies prior to actual deployment is common in research.<sup>3</sup>

## 2.2 | Extensible cloud simulators

Cloud simulators provide a system model to represent a cloud environment. For example, the CloudSim toolkit follow a system model consisting of cloud brokers and data centers.<sup>4</sup> Another example is the GreenCloud simulator, which focuses on network and energy and has a system model consisting of data centers and network devices with their energy models.<sup>18</sup> In most simulators, the system models are extensible, meaning they provide extension points to inject scenario-specific logic. Users can leverage these extension points to experiment with novel approaches, such as scheduling policies and allocation policies.<sup>2</sup>

Furthermore, the simulation platform itself can be extensible. This is the case for widely used simulators like the CloudSim toolkit, which allows for reusable code,<sup>4</sup> and enables the creation of extended simulators that are use case specific.<sup>2</sup> For example, CloudSimSDN provides support for Software Defined Networks in CloudSim by extending the vanilla CloudSim toolkit.<sup>19</sup> Another example is GreenCloud, which extends the Network Simulator NS2<sup>20</sup> and adds cloud environment components to create an energy and network-focused simulator.<sup>18</sup>

Therefore, the extensibility of cloud simulators, which involves providing extension points and the ability to extend itself, is significant in accommodating various cloud simulation scenarios. Most simulators leverage programming language features, such as object-oriented representation of components, to implement extensibility.<sup>4,21</sup> However, these implementations often come with a steep learning curve. At the same time, simplifying the cloud environment system models (e.g., using script-based system models) to reduce the learning curve can compromise extensibility, as they encapsulate the object-oriented representation of inner components.<sup>3,12</sup>

## 2.3 | CloudSim toolkit

The CloudSim toolkit is a Java-based event-driven simulation toolkit for Cloud environments.<sup>4</sup> It is widely used in research<sup>10,11</sup> due to its rich features and its ability to be extended and customized to cater to most simulation scenarios. CloudSim supports the virtualization of physical machines, offering customizable provisioning policies for virtual machines (VMs). This capability extends to modeling and simulating application containers. Moreover, CloudSim's power packages facilitate the modeling of the energy aspects of computations, thereby enabling the simulation of energy-optimization techniques. CloudSim evolves as the field advances enabling researchers to experiment with novel concepts. Such examples features are Microservices architectures, Edge/Fog computing,<sup>22</sup> Serverless computing,<sup>23</sup> and Quantum computing.<sup>24</sup> The architecture of CloudSim allows for dynamic insertion of components, enhancing the management of the simulation lifecycle with features like stop-and-resume. The simulation environment's interconnected networks can be effectively modeled using network topologies. This setup allows for an analysis of the impact of network characteristics, such as latency and bandwidth, on the communication between components. Furthermore, CloudSim includes user-defined policies for resource management, which can be tailored to various customized scenarios, such as allocating VMs to host resources or assigning processing elements to active VMs in a granular manner. This level of customization reduces the effort required for system model development, enabling researchers to concentrate more on aspects of resource management.

The CloudSim toolkit has garnered significant attention within the scientific community, continuously fostering a rich ecosystem of extensions. One notable extension is WorkflowSim, developed by the University of Southern California, which adds a higher layer of workflow management to CloudSim. This layer addresses heterogeneous system

overheads and failures,<sup>25</sup> enabling effective evaluations of scientific workflows in distributed settings. A distinctive aspect of modern cloud infrastructures is the investigation of cloud storage and its energy consumption. In this context, CloudSimDisk<sup>26</sup> proposes a scalable module for CloudSim, extending its capabilities to support energy-aware storage solutions. This module has been validated by comparing simulation results with analytical models designed to gauge the energy consumption of hard disk drives in cloud systems. NetworkCloudSim addresses the lack of advanced application models in heterogeneous clouds. It focuses on scalable networking and a generalized application model, encompassing applications such as message passing and workflows. This allows for more accurate evaluations of scheduling and resource provisioning policies.<sup>27</sup> The migration of software systems to the cloud necessitates the evaluation of competing cloud deployment options (CDOs). To this end, CDOSim offers a tool for assessing the cost and performance properties of CDOs,<sup>28</sup> integrating a migration framework with CloudSim and leveraging provider-specific performance characteristics.

Furthermore, CloudSimEx aims to maintain a collection of extensions that are not only research-oriented but also focused on improving software engineering aspects. These improvements include enhanced logging utilities, the generation of CSV files for analysis purposes, and the capability to run multiple experiments in parallel.<sup>29</sup> Cloud2Sim concentrates on executing CloudSim simulations in a distributed manner, utilizing resources such as multi-core CPUs.<sup>30</sup> It achieves this through a distributed object storage approach. CloudSimSDN<sup>19</sup> introduces software-defined networking (SDN) modeling to CloudSim simulations, enabling researchers to explore SDN-oriented resource management approaches. The diverse ecosystem of CloudSim extensions highlights CloudSim's ability to offer a versatile simulation framework for cloud computing.

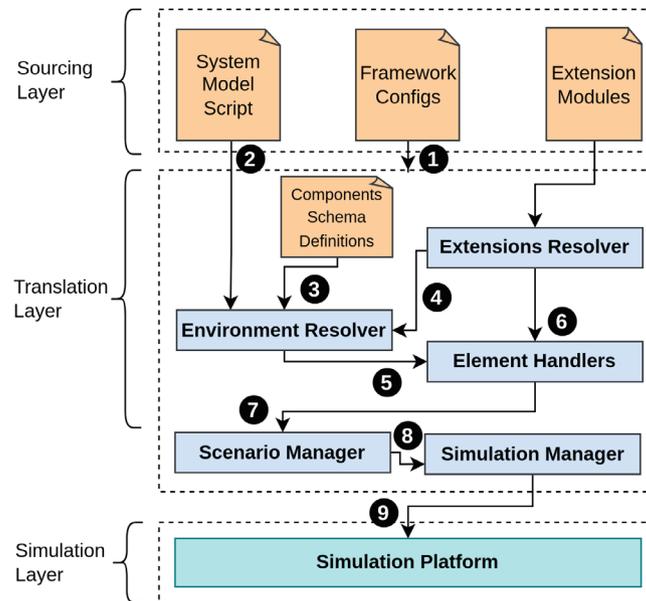
At the same time, the CloudSim toolkit, and most of its extensions are based on modules that are written in java, which means that its simulations are developed as Java projects. Therefore, an additional overhead of programming language expertise is added to the development life cycle. Besides, CloudSim does not specify a standard pattern for implementing simulation scenarios, making it complex to reuse common components across simulations. This also leads to simulation code bases that are difficult to manage and have poor code readability. Aforementioned drawbacks are primarily caused by the lack of a standard framework for developing modularized simulations and the absence of a simplified representation of the simulation system model, such as a human-readable script. A modularized framework would enable CloudSim users to share common simulation components, drastically improving code readability and maintainability. A simplified representation of the simulation system model would allow the development of simplified simulations using a top-down approach without requiring expertise in a programming language.

To that end, existing literature aim to minimize the programming language overhead in CloudSim simulations. These works focus on providing a graphical user interface (GUI) to facilitate the design and execution of simulations through CloudSim.<sup>3,12</sup> They enable users to set configurable parameters via the GUI and observe statistics of the ongoing simulation. The underlying implementation is responsible for translating between the GUI and CloudSim. A similar approach can be implemented using a human-readable script.<sup>11</sup> This method eliminates the need for users to have proficiency in a programming language, as the script and its structure for modeling system components are designed to be human-readable.

However, while approaches aimed at reducing programming overhead can streamline the use of CloudSim, they may also limit the toolkit's extensibility. Extensibility is an important feature of CloudSim, forming the foundation of its extensive ecosystem of extensions. Compromising on this aspect diminishes the ease with which implemented approaches can be integrated into existing extensions. For instance, DartCSim, which offers a GUI interface, permits the overriding of only a limited set of methods,<sup>11</sup> thus constraining the scope for customization. Similarly, CloudSim-Plus Automation, which employs a human-readable script to reduce programming complexity, restricts users in customizing implementations due to its specific usage of the Datacenter component.<sup>11</sup> Addressing this gap, our work introduces a novel framework that significantly reduces programming overhead while minimally impacting CloudSim's extensibility.

### 3 | ARCHITECTURAL FRAMEWORK

In this section, we propose a generic modularized architectural framework for designing the Cloud environment simulation system model using a human-readable script that can be automatically translated into a concrete simulation implementation.



**FIGURE 1** Architectural framework for script-based simulations.

Figure 1 illustrates the proposed architectural framework. Our architecture consists of three main layers: the *Sourcing Layer*, which maintains user-modifiable files and configurations; the *Translation Layer*, which handles the logic of translating the *System Model Script* into an actual simulation; and the *Simulation Layer*, which represents the simulation platform. The following sections provide a detailed description of each layer and explain the overall control flow for translating the *System Model Script* into a concrete simulation.

### 3.1 | Sourcing layer

The *Sourcing Layer* provides the following external sources.

- **System Model Script:** The simulation system model is described using a human-readable scripting language. The system model comprises multiple interconnected components and can be simplified using a top-down approach. For instance, a regional Cloud zone itself is a system model that includes one or more Datacenters. Each Datacenter consists of a set of hosts, and these hosts, in turn, have associated processing elements. Thus, the *System Model Script* describes each of these components and aggregates them into an overall system model component.
- **Framework Configurations:** Provides various configurations required to initialize the framework, such as location information of the files in the *Sourcing Layer*.
- **Extensions Modules:** Provides modularized extensions to the user. These extensions are used to customize the simulation platform, as well as the framework itself.

### 3.2 | Translation layer

This layer consists of the following modularized components to translate the human-readable system model into a concrete simulation.

- **Environment Resolver:** Parses the *System Model Script* file, into a set of components that describes the system model. It uses the supplied component schema to understand the *System Model Script*. Upon parsing, components are aggregated into an overall system model component and handed over to the corresponding *Element Handler*. For example,

a host described in the *System Model Script* can contain 7 processing elements. This is parsed into a host component having its processing elements attribute set to 7. Afterwards, multiple hosts components are aggregated to create the Datacenter system model, which is then handled by a corresponding *Element Handler*.

- **Element Handler:** Processes a component in the simulation environment, and provides the corresponding simulation logic that needs to be implemented. Therefore, an *Element Handler* is implemented per Simulation Platform, and injected into the *Translation Layer* as an extension.
- **Extensions Resolver:** Provides a central resolver to materialize objects from the provided extension modules. It generates extensions for various *Element Handlers*, *Translation Layer* modules, and the extensible components in the *Simulation Platform*.
- **Scenario Manager:** Handles simulation scenario by executing *Element Handler* components.
- **Simulation Manager:** Manages the overall simulation, such as initializing the simulation platform, initiating the system model translation, starting the simulation and overseeing its execution.

### 3.3 | Simulation layer

This layer consist the simulation platform. The *Translation Layer* interacts with it during the initialization and simulation execution stages. It decouples the simulation platform logic from the script-based translation logic, thereby allowing framework to be implemented with different simulation platforms.

### 3.4 | System model script to simulation translation flow

Figure 1 denotes the following control flow of the architectural framework.

1. ❶ initialize the *Translation Layer*. This includes registering available *Element Handlers*, initializing *Extensions Resolver* with the location of extension modules and so forth.
2. ❷ and ❸ denotes *Environment Resolver* parsing the system model. It reads the *System Model Script* and constructs the system model component using the schematics of the components provided with ❸. Upon construction of the system model component, its associated *Element Handler* is materialized via the *Extensions Resolver* in ❷, and then parsed system model component is handed over.
3. ❹, ❺, and ❻ denotes *Scenario Manager* handling the simulation scenario using the system model element handler. In doing so, it materializes other associate *Element Handlers* of the system model element handler in a top-down approach.
4. ❼ and ❽ denotes *Simulation Manager* managing the simulation by executing the simulation scenario from ❽, with the *Simulation Platform* provided by ❹.

## 4 | IMPLEMENTATION

In this section, we describe the implementation of our proposed framework for CloudSim toolkit.

### 4.1 | System model script and components schema definitions

To simplify the Cloud environment system model with a human-readable script, we choose the human-friendly data serialization language, YAML (YAML Ain't Markup Language).<sup>16</sup> YAML provides the capability to describe multiple components and then to aggregate the system model. Besides, YAML is widely used in the industry from managing configurations,<sup>31</sup> to describing APIs, thereby provides a familiar experience to users.

To define the schema to write components in the *System Model Script*, we use OpenAPI 3.0 specification.<sup>32</sup> OpenAPI 3.0 is a machine-readable interface definition language and it is widely used with web services.<sup>33</sup> It provides specifications to describe object schematics in a top-down approach.

```

Datacenter:
  type: object
  properties:
    variant:
      $ref: '#/components/schemas/Extension'
  characteristics:
    $ref: '#/components/schemas/Datacenter...'
  vmAllocationPolicy:
    ...

```

Listing 1: Component Schematics

```

@Generated(value = "org.openapi...")
public class Datacenter {

    @JsonProperty("variant")
    private Extension variant;

    @JsonProperty("characteristics")
    private DatacenterCharacteristics cha...
    ...

```

Listing 2: Generated Java Class

```

Datacenter: *Datacenter
  variant:
    className: "org.cloudbus.cloudsim.Datacenter"
  characteristics: *Characteristics
  vmAllocationPolicy:
    className: "org.cloudbus.cloudsim.VmAllocation..."
  storage: ""
  schedulingInterval: 0
  ...

```

Listing 3: System Model Script

Firstly, we use OpenAPI code generation library during *CloudSim Express* compilation time to read components schema definitions and convert them to plain-old-java-object (PoJo) classes. During its run time, *CloudSim Express* reads the *System Model Script* YAML file and parse YAML components to objects of aforementioned PoJo classes. Both compilation time and run time translations of the Datacenter component are demonstrated with Listing 1,2,3 and Figure 2. In which, Listing 1 shows the object schematics of the Datacenter component, alongside the corresponding attribute names and the type of the value complying with OpenAPI 3.0 specification. Listing 2 shows the PoJo class generated upon compilation of the *CloudSim Express*. Listing 3 shows the datacenter information sourced from the *System Model Script*, for a given simulation scenario. This information is written according to the schematics defined in Listing 1. As shown in Figure 2, this information is then translated into a PoJo object during *CloudSim Express* run time.

```

Expression:
.globalDatacenterNetwork).systemModelLDT0).zones).get(0)).d_*
Use Ctrl+Shift+Enter to add to Watches

Result:
∞ result = {Datacenter@2814} "class Datacenter {\n  variant: class Extension {\n... View
  variant = [Extension@2820] "class Extension {\n  className: org.cloudbus... View
  > className = "org.cloudbus.cloudsim.Datacenter"
  > copies = null
  > extensionProperties = null
  > characteristics = {DatacenterCharacteristics@2821} "class DatacenterChai... View
  > vmAllocationPolicy = {Extension@2822} "class Extension {\n  className: ... View
  > className = "org.cloudbus.cloudsim.VmAllocationPolicySimple"
  > copies = null
  > extensionProperties = null
  > storage = ""
  > schedulingInterval = {Integer@2824} 0
  > value = 0
  > name = "regional-datacenter"

```

FIGURE 2 Runtime parsed datacenter object.

## 4.2 | Environment resolver

The Section 4.1 describes how *CloudSim Express* translates *System Model Script* to a system model PoJo object. The runtime translation process of that (also demonstrated with Listing 3 and Figure 2) is implemented with the *Environment Resolver* component. It uses a YAML parsing library to read the *System Model Script* and map the parsed information into the PoJo object using an object mapping library. Afterwards, it traverses through registered Element Handlers and materializes a new matching Element Handler for the system model PoJo object.

## 4.3 | Extensions resolver

An extension in *CloudSim Express* is a java module, packed as a jar. Extensions can extend the CloudSim toolkit, as well as the *CloudSim Express* framework itself. The *Extensions Resolver* module implementation provides APIs to materialize extension objects. During the launch of *CloudSim Express*, it reads and loads the class files packed in extension jar modules. It then provides APIs to allow other modules in the *Translation Layer* to invoke extension creation requests, with necessary information provided as custom java constructor arguments. We use java reflections feature to create objects using the loaded extension classes and the provided constructor arguments.

## 4.4 | Segregating CloudSim simulation logic into element handlers

A standard CloudSim simulation involves building the system model in a java method body. We segregate this implementation to a series of recursive calls of Element Handler components. Each Element Handler provides implementation logic to a specific component in its *handle()* method, and for any other associated components, it makes a recursive method call of the same method of the corresponding Element Handler. This flow is demonstrated with Figure 3. In which, a regional zone handler require a datacenter object for its implementation logic. It then calls *handle()* method of the datacenter handler and obtain the required datacenter object. The datacenter handler requires a host object for its implementation logic, which then is obtained by calling the same method of the host handler. The actual control flow for this scenario is depicted by the number ❶ to ❹.

Overall, the *handle()* method instantiates the required CloudSim classes and creates the elements that are required. A series of element handler calls then generate all the required instances of CloudSim classes to establish the simulation environment. These element handlers can be easily customized to meet user requirements through the schema definition file. In this file, users define the schema for new elements and update the framework with corresponding POJO (Plain Old Java Object) files, which is automated during the build process. Subsequently, the corresponding element handler implementations are dynamically injected into the framework.

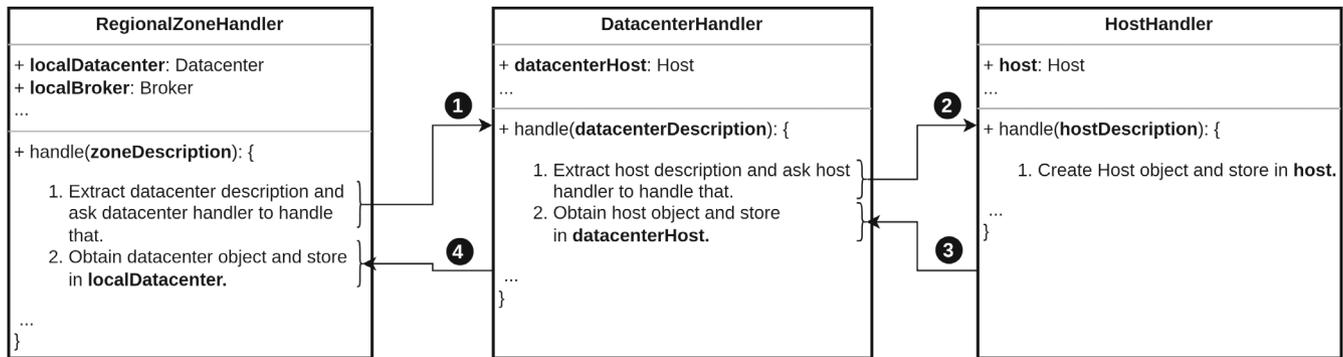


FIGURE 3 Segregation of CloudSim simulation logic with element handlers.

## 4.5 | Execution of the simulation

Once the Element Handler of the system model is realized (i.e., the process discussed in Section 4.2), it is then passed to the Scenario Manager. The Scenario Manager is controlled by the Simulation Manager. When *CloudSim Express* starts executing, the Simulation Manager instructs Scenario Manager to build the simulation scenario. The Simulation Manager achieves this by calling the *handle()* method of the system model Element Handler. As explained in Section 4.4, this eventually implements the CloudSim logic for building the system model. Afterwards, the Simulation Manager starts the simulation with CloudSim platform and oversee towards its completion.

## 4.6 | Managing extensions

Being able to extend is a powerful feature of CloudSim. Our proposed platform provide the flexibility to keep that intact, by integrating CloudSim extensions with *Extensions Resolver*. We define an attribute *variant*, in the *System Model Script* to indicate that a custom version of a component needs to be used. Listing 3 and Figure 2 demonstrate an example of this. Which, the datacenter component that needs to be used is of the type *org.cloudbus.cloudsim.Datacenter*. This block of information is passed to the relevant element handler, and the element handler materializes a datacenter component of the mentioned type via the *Extensions Resolver*, through the java reflection feature. The user needs to place the extension class packed into a jar in the extensions location, and set the corresponding class name in the *System Model Script* file. Afterwards, the *CloudSim Express* tool is able to interpret the relevant extended version.

Apart from the simulation platform, the components in the *CloudSim Express* framework can also be extended in the same manner. However, since this is not a part of the simulation, the user is required to place the corresponding jar same as before, but the corresponding class name needs to be configured in the configuration file of *CloudSim Express*. This feature is useful if a user wants to change the default behaviour of a component in the *Translation Layer*, such as changing the behaviour of a default Element Handler.

## 4.7 | Generalizability

Apart from the *CloudSim Express* implementation, the proposed framework is general enough to be applicable to any other simulator that builds the simulation system model using a top-down approach. In such scenarios, the implementation logic specific to the simulator is customized in the *Element Handler* components, and the *Translation Layer* components are customized as needed. Each of these components is designed to be pluggable, making the deployment of customized components straightforward.

## 5 | PERFORMANCE EVALUATION

To evaluate the effectiveness of our framework, we implement a use case with *CloudSim Express* and baselines. We collect qualitative metrics and quantitative aspects of the development life cycle productivity.

## 5.1 | Design of the use case

CloudSim simulations can range from experimenting with allocation policies to experimenting with novel infrastructure changes, such as evaluating low power datacenters. These scenarios are implemented by extending the CloudSim toolkit. Therefore, in order to evaluate true productivity improvements from *CloudSim Express*, the use case should stress extensibility of the CloudSim toolkit.

The standard extension points of the CloudSim toolkit support a variety of policies, such as virtual machine allocation. In addition to that, it can be further extended using its object-oriented architecture. To capture both extensible aspects, we design following use case requirements.

- **R1:** A datacenter that is aware of its Virtual Machines (VMs) allocation to physical hosts.
- **R2:** A datacenter that periodically monitors its Virtual Machines (VMs)

The CloudSim toolkit utilizes the abstract Java class *VmAllocationPolicy* to define the logic for allocating VMs to physical machines during the simulation. By default, CloudSim includes an implementation of the worst-fit algorithm (i.e., allocating a VM to the host with the most processing elements) through the concrete class *VmAllocationPolicySimple*, which extends the aforementioned abstract class. In practical scenarios, users can implement their own VM allocation algorithms by extending the same abstract class. Therefore, **R1** is also implemented by creating a new class that extends the *VmAllocationPolicy* abstract class. Once the implementation is completed, the CloudSim simulation code needs to be modified to include the newly created allocation policy class, and the codebase must be recompiled before running the simulation.

The CloudSim extension points do not cover all stages of the simulations. Anything that is not covered by the standard simulation points is implemented through the object-oriented design of the CloudSim toolkit. In such scenarios, users extend specific classes that represent the components in the simulation system model. Therefore, **R2** is implemented by extending the standard CloudSim class "Datacenter" and overriding the method "updateCloudletProcessing". Within the overridden method, the logic corresponding to periodic virtual machine monitoring is implemented. Afterward, the extended "Datacenter" class needs to be included in the CloudSim simulation code, and the codebase must be recompiled before running the simulation.

In both implementations, users are required to be familiar with the CloudSim simulation codebase and need to invest effort into recompiling the codebase multiple times. Overall, this process involves significant effort. However, it offers users fine-grained flexibility in designing the simulation. Therefore, an ideal approach for reducing the aforementioned programming language overhead should also involve minimal compromise in simulation design flexibility.

## 5.2 | Baseline approaches

We use the following baseline approaches that reduces the programming language overhead of the CloudSim toolkit.

- **CloudSim Plus Automation:**<sup>11,34</sup> A YAML (YAML Ain't a Markup Language) script-based simulation approach for CloudSim simulations.
- **GITS: Generic Input Template for Cloud Simulators:**<sup>10</sup> A JSON (JavaScript Object Notation) script-based generic input template based approach for CloudSim simulations.

## 5.3 | Implementation with CloudSim plus automation

The CloudSim Plus Automation provides YAML script-based system model design. We implemented the use case by designing a single Datacenter, with a single customer. However, the CloudSim Plus Automation does not have native flexibility for custom extensions. It only supports the standard extension points supported with the CloudSim Plus project,<sup>21</sup> which is an enhanced version of the CloudSim toolkit. Therefore, to implement **R1** and **R2** from Section 5.1, we follow the approach below after inspecting the source code of CloudSim Plus Automation.

- **R1:** Write an extended VM allocation policy class, *VmAllocationPolicyCustom*, using the java package, *org.cloudsimplus.allocationpolicies*. This is because CloudSim Plus Automation only supports allocation policy classes with the naming prefix *VmAllocationPolicy*, and residing in the aforementioned java package. When a custom class is implemented in this manner, it can be selected via the YAML script.
- **R2:** Write an extended *DatacenterSimple* class. The *DatacenterSimple* class represents datacenters in CloudSim Plus Automation. In order to select it, we modified the source code. Because selecting a custom datacenter class is not

```
datacenters:
  - !datacenter
    amount: 1
    vmAllocationPolicy: Custom
    ...
  sans:
    - !san
      ...
  hosts:
    - !host
      amount: 5
      ...
```

Listing 4: CloudSim Plus Automation: YAML Script

```
{
  "name": "GITS template",
  ...

  "DC": {
    "name": "RegionalDC",
    ...

  "Rack": {
    ...

  ...
```

Listing 5: GITS: JSON Input Template

```
- <<: *Host
  ...
  copies: 5
Characteristics: &Characteristics
  ...
Datacenter: &Datacenter
  variant:
    className: "org.cloudbus...
  characteristics: *Character...
```

```

vmAllocationPolicy:
  className: "org.cloudbus...
...

```

Listing 6: CloudSim Express: System Model Script

To load custom classes during runtime, both were implemented in the CloudSim Plus Automation project. Afterward, CloudSim Plus Automation project was recompiled to obtain a modified tool with custom classes. Finally, the tool was executed with the corresponding YAML script, which is denoted in Listing 4. In this script, we set VM allocation policy value as *Custom*, so that our custom allocation policy implementation is selected.

## 5.4 | Implementation with GITS

The GITS (generic input template for cloud simulators) focuses on representing the Cloud environment with either textual (JSON (JavaScript Object Notation) template) or graphical representation, and for this comparison, we use the former. The GITS for CloudSim provides a JSON template describing the Cloud Environment, and a GITS library file to convert Datacenter and hosts information from the JSON template to the GITS java objects.

The Figure 4 depicts the UML diagram for the objects in the JSON template corresponds to the use case. We implemented GITS library file with the logic to handle objects in the template.<sup>10</sup> Once GITS java objects are created, the CloudSim Datacenters and hosts are populated based on their values. Therefore, we implemented the use case requirements with the standard CloudSim simulation described in Section 5.1 while using GITS as a layer to interpret datacenter and host values from the JSON input template, which is denoted in Listing 5.

## 5.5 | Implementation with CloudSim express

The system model of *CloudSim Express* implementation includes a regional zone with a datacenter. The extensions required by **R1** and **R2** from Section 5.1 are implemented as follows.

- **R1:** Write an extended VM allocation policy class
- **R2:** Write an extended *Datacenter* class

Both custom classes were developed in a separate java project, and compiled into a jar file. The resulting jar file is then copied to the *CloudSim Express* tool.

Listing 6 denotes *System Model Script* for the use case, in which we modified it to select the two custom classes. This *System Model Script* is also copied to the tool. Since default element handlers follow the commonly used cloud

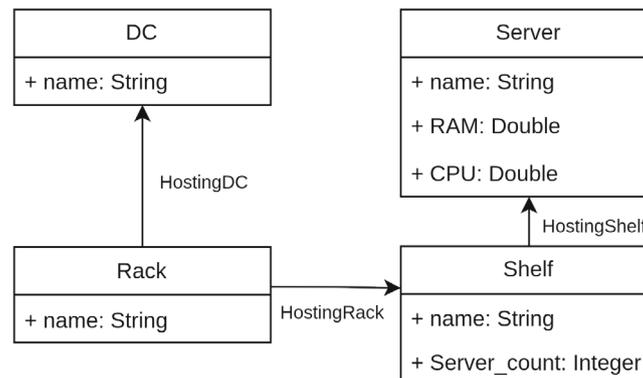


FIGURE 4 GITS UML diagram for generic input template.

architectures, we do not need to modify them. Therefore, we executed *CloudSim Express* tool, which reads the script and load the custom classes during startup, and then to execute the simulation. *CloudSim Express* tool require least effort in deploying custom simulation scenarios, as the tool has automated most of the class loading procedures.

## 5.6 | Productivity analysis

Productivity can be measured in two aspects: quantitative and qualitative. We measure the quantitative aspect by considering the amount of programming involved. The primary goal of a scripted simulation is to minimize the time spent on coding. Table 1 depicts the differences in productivity between *CloudSim Express* and the baseline approaches for the implemented use case.

We measure the lines of code to gauge the project's magnitude. Additionally, we measure the complexity of the code because customizing the use case sometimes requires analyzing and modifying the simulation platform code. If the code is more complex, it takes more time to analyze. Since all approaches are Java-based solutions, we utilize JaCoCo, an open-source code coverage report generation tool,<sup>35</sup> to measure the lines of code and code complexity. JaCoCo calculates cyclomatic complexity value for each nonabstract method in the code, which is an indication of number of unit tests required for possible paths through the methods. A higher complexity number require increased effort in analyzing the control flow of the code. For both measurements, we excluded the GITS library file implementation code from Section 5.4 as it can be imported as a library.

We measure four qualitative aspects: framework recompilation, CloudSim implementation, runtime extension injecting, and extending via script support. Framework recompilation indicates whether modifying the provided framework is required to achieve the customized behavior. CloudSim implementation indicates whether at least a partial CloudSim simulation code implementation is needed. Runtime extension injecting indicates whether extended components are dynamically provided without recompiling the framework. Extending via script support indicates whether framework extendability is achieved via the *System Model Script*.

The quantitative values of the approaches are denoted in Table 1. The CloudSim Plus Automation involves the largest code base in terms of both code complexity and lines of code. This is because the required customization in the use case mandates analyzing the framework code base and identifying how custom classes can be incorporated in the best possible manner, and then recompiling the framework with customization. In comparison, GITS involves a relatively smaller code base. This is because the script-to-java translation is performed via the provided library file, which can be used out of the box. The library file is then consumed in the standard CloudSim simulation implementation, resulting in a relatively smaller code base overall. Among the three approaches, *CloudSim Express* provides the smallest code base as it only requires implementing the custom classes. The extension injecting is done dynamically without involving coding implementations.

The qualitative values of approaches are also denoted in Table 1. The CloudSim Plus Automation needs to be modified to incorporate the customized classes; thus, it does not support runtime extension injecting, and a framework recompilation is also needed. However, it does not require CloudSim simulation implementation, as the system model is generated via the provided script with extension support, and the simulation lifetime is managed via the framework itself. On the other hand, GITS performs worst in all qualitative aspects. This is because it only supports converting the script information to Java objects. The rest of the simulation has to be implemented via CloudSim in a standard manner. The *CloudSim Express* provides the best development experience since its dynamic extension injecting feature only requires providing the extension classes. The rest of the simulation is managed automatically.

TABLE 1 Productivity comparison of scripted simulation approaches.

Approach	Quantitative		Qualitative			
	JaCoCo: Code complexity	JaCoCo: Lines of Code	No framework recompilation	No CloudSim implementation	Runtime extension injecting	Extending via script
CloudSim plus automation	314	740	✗	✓	✗	✓
GITS (Excluding library code)	14	104	✗	✗	✗	✗
CloudSim Express	4	11	✓	✓	✓	✓

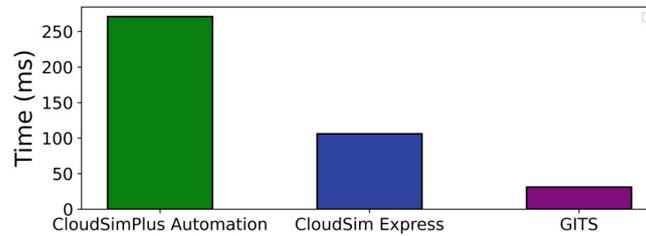


FIGURE 5 Comparison of processing overhead times.

Figure 5 compares the processing overhead times for each approach when handling the *System Model Script*. For the GITS approach, this duration was measured from the tool's launch time to the initiation of the CloudSim simulation, thus encompassing the time taken to process the script and convert it into the model. The method for calculating time is identical for *CloudSim Express*, covering both script processing and framework loading times. However, the latter tends to be lengthier compared to GITS. In contrast, for CloudSimPlus Automation, the measured duration extends from the tool's launch to the invocation of the CloudSimPlus framework, which is the lengthiest of the three. Notably, all approaches incur a processing delay of less than a quarter of a second before commencing the actual simulation, indicating that the impact of the *System Model Script* processing time is generally minimal. Compared to the baseline approaches, *CloudSim Express* exhibits a reasonable overhead time.

To summarize, we observed that *CloudSim Express* outperforms the baselines in terms of productivity, both qualitatively and quantitatively. We evaluated all approaches using a practical simulation use case that emphasized the need for extensibility, which is a prominent feature of the CloudSim toolkit. CloudSim Plus Automation offers a reasonably good developer experience by automating the simulation life cycle through a *System Model Script* that supports the configuration of different policies at runtime. However, implementing extensions requires managing a relatively large code base and necessitates platform recompilation. On the other hand, GITS involves a smaller code base but provides only partial automation of the simulation life cycle. *CloudSim Express* stands out by providing dynamic extension capabilities while automating the simulation life cycle, having a reasonable processing time overhead. In addition to its developer-friendly qualitative aspects, it also achieves a reduction of 71.43% in code complexity and 89.42% in lines of code compared to the baselines.

## 6 | RELATED WORK

We investigate the literature for approaches that reduce programming language overhead in developing simulated Cloud computing experiments. Cloud simulators are usually bound to a specific programming language.<sup>4,6</sup> Due to that, expertise in programming languages is needed before implementing a simulation scenario, which is an additional overhead. Instead, most approaches offer either textual (in the form of a script) or graphical modeling of Cloud environments,<sup>3,9,11-15</sup> or both.<sup>10</sup> Most such approaches allow implementing end-to-end simulations without coding involved, or with a reduced amount of code.<sup>10</sup> Another approach attempts to reduce the effort required for developing scaling simulations by adopting IEEE standards for interoperability.<sup>36</sup> However, it still requires considerable programming effort to implement the scenario. Another proposal suggests simplifying existing simulation approaches by providing flexible combinations of different functional components.<sup>37</sup> However, this also requires considerable programming effort. Productivity can also be improved by the manner that an approach is deployed. There are two primary approaches used in the literature, as a tool and as a library. Most approaches are implemented in the form of a tool,<sup>3,11-15</sup> which either a Graphical User Interface (GUI) based<sup>12</sup> or a command line based.<sup>11</sup> Library-based approaches<sup>9,10</sup> still require partial coding implementation. Alongside that, some approaches are tightly coupled to the underlying simulator. Such implementations mandate the use of a specific simulator implementation.<sup>10,14,15</sup> In contrast, approaches offered as a separate component from the simulator provide reduced effort in customization.<sup>3,9,11-13</sup> Our proposed approach opts for maximum ease of use, thus implementing a tool that consumes textual representation of the system model. Its generic architecture is implemented as a decoupled layer providing room for customisation.

Simplified simulation approaches tend to encapsulate internal simulation details, thus often compromising in customization. For example, in the textual or graphical representation of the system model, only a limited set of Cloud

TABLE 2 Comparison of related work.

	Human-readable cloud environment modeling	Programming language syntax	Deployment mode as a tool	De-coupled from simulation platform	Runtime extension injection	Native framework extensibility support	Introducing/modifying environment components
EPCSAC <sup>14</sup>	✓	✗	✓	✓	✓	✗	✗
VFCSIM <sup>9</sup>	✓	✗	✗	✗	✗	✗	✗
iCanCloud <sup>13</sup>	✓	✗	✓	✗	✗	✗	✗
CloudReports <sup>15</sup>	✓	✗	✓	✓	✓	✗	✗
DartCSim <sup>3</sup>	✓	✗	✓	✗	✓	✗	✗
CloudAnalyst <sup>12</sup>	✓	✗	✓	✗	✗	✓	✗
GITS <sup>10</sup>	✓	Partial	✗	✓	✗	✗	✗
CloudSim plus automation <sup>11</sup>	✓	✗	✓	✗	✗	✗	✗
CloudSim express (this work)	✓	✗	✓	✓	✓	✓	✓

environment elements are available.<sup>13,15</sup> Introducing a new element, or modifying an existing element involves a complex development effort that often requires framework recompilation. For example, EdgeSimPy, a python based edge computing simulator simplifies scenario definitions via a json input file, however, users are constrained within its schema,<sup>38</sup> therefore granular changes require modifications in the simulator code. Some approaches allow partial customization via mandatory extension points in the simulator platform, such as allocation policies in CloudSim,<sup>4</sup> providing partial dynamic extension capabilities. For example, DartCSim<sup>3</sup> allows users to dynamically inject extensions without a framework recompilation by providing an embedded code editor. However, users are limited to the provided set of methods. CloudReports<sup>15</sup> further enhance this by allowing users to inject extensions that are developed separately. It still requires providing a separate XML file, which is an overhead. Apart from that, a framework customization might required. For example, the vanilla implementation of CloudSim Plus Automation<sup>11</sup> does not support allocation policies in a custom Java package, which requires an extension of the framework. In general, framework extensibility is rarely supported (one such case is CloudAnalyst,<sup>12</sup> which supports framework extensibility in its architecture), thus requiring complex code analysis. Our proposed approach supports extensibility in its design by providing a common extension consumption approach for both extension points in the simulator, and the framework itself. Its generic design with handlers for Cloud environment components provides native support for new Cloud environment components, as well as modifying existing ones.

Table 2 depicts comparison amongst the related work, and the proposed approach. Most approaches significantly compromise between simplified ease of use, and ability to customize. Our proposed approach is designed from its architecture to jointly improve both of those aspects.

## 7 | CONCLUSIONS AND FUTURE WORK

Cloud computing environment simulators enable the experimentation of novel infrastructure designs and management approaches with significantly less time, cost, and effort. However, they are tightly coupled to programming language ecosystems, requiring effort in designing, configuring, and programming language expertise. Existing work aims to reduce this overhead but often compromises on simulator extensibility. In this work, we propose an architectural framework for Cloud environment simulators to realize a script-based simulation, aiming to reduce the overhead of programming language expertise while minimizing compromises on extensibility. We implemented the proposed framework for the widely used Cloud simulator, the CloudSim toolkit, and compared it against state-of-the-art baselines for a practical use case. Our evaluations show that *CloudSim Express* achieves extensible simulations with minimal code compilations and surpasses the baselines with over a 71.43% reduction in code complexity and an 89.42% reduction in lines of code.

**Future Work:** The proposed framework segregates simulation logic through a series of handlers. We plan to further reduce the inter-dependency of these handlers to achieve even more isolated implementation logic. Additionally, we intend to develop a graphical user interface using the *System Model Script*. Furthermore, our proposed framework can be extended to integrate powerful scripting languages other than YAML, such as LUA, to enable even greater flexibility in managing extensions.

### AUTHOR CONTRIBUTIONS

**Tharindu B. Hewage:** Conceptualization, Data curation, Methodology, Software, Validation, Writing—original draft. **Shashikant Ilager:** Supervision, Methodology, Writing—review & editing. **Maria A. Rodriguez:** Supervision, Methodology, Writing—review & editing. **Rajkumar Buyya:** Conceptualization, Supervision, Methodology, Visualization, Writing—review & editing.

### DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available at <https://github.com/Cloudslab/cloudsim-express> alongside the release of *CloudSim Express* as an open-source tool under the GPL-3.0 license. <https://github.com/Cloudslab/cloudsim-express> CloudSim Express will be a part of the upcoming release of CloudSim 6.0.

### ACKNOWLEDGMENT

Open access publishing facilitated by The University of Melbourne, as part of the Wiley - The University of Melbourne agreement via the Council of Australian University Librarians.

## ORCID

Tharindu B. Hewage  <https://orcid.org/0000-0001-6348-4602>

Rajkumar Buyya  <https://orcid.org/0000-0001-9754-6496>

## REFERENCES

- Smith D. The future of cloud computing in 2027: From technology to business innovation. 2023. <https://www.gartner.com/en/doc/768816-the-future-of-cloud-computing-in-2027-from-technology-to-business-innovation>
- Mansouri N, Ghafari R, Zade BMH. Cloud computing simulators: A comprehensive review. *Simul Model Pract Theory*. 2020;104:102144.
- Li X, Jiang X, Huang P, Ye K. DartCSim: An enhanced user-friendly cloud simulation system based on CloudSim with better performance. *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*. IEEE; 2012:392-396.
- Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw Pract Exp*. 2011;41:23-50.
- Malik AW, Bilal K, Aziz K, et al. CloudNetSim++: A toolkit for data center simulations in OMNET++. *Proceedings of the 2014 11th Annual High Capacity Optical Networks and Emerging/Enabling Technologies (Photonics for Energy)*. IEEE; 2014:104-108.
- Fernández-Cerero D, Fernández-Montes A, Jakóbič A, Kołodziej J, Toro M. SCORE: Simulator for cloud optimization of resources and energy consumption. *Simul Model Pract Theory*. 2018;82:160-173.
- Sadowski C, Söderberg E, Church L, Sipko M, Bacchelli A. Modern code review: A case study at Google. *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. Association for Computing Machinery; 2018:181-190.
- Prathanrat P, Polprasert C. Performance prediction of Jupyter notebook in JupyterHub using machine learning. *Proceedings of the 2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*. IEEE; 2018:157-162.
- Tsakanikas V, Dagiuklas T. VFCSIM: A simulation framework for real-time multi-service Virtual Function Chains deployment. *Proceedings of the GLOBECOM 2022–2022 IEEE Global Communications Conference*. IEEE; 2022:2644-2649.
- Jammal M, Hawilo H, Kanso A, Shami A. Generic input template for cloud simulators: A case study of CloudSim. *Softw Pract Exp*. 2019;49:720-747.
- Silva Filho MC, Rodrigues JJPC. Human readable scenario specification for automated creation of simulations on CloudSim. *Proceedings of the Internet of Vehicles – Technologies and Services*. Springer, Cham; 2014:345-356.
- Wickremasinghe B, Calheiros RN, 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*. IEEE; 2010:446-452.
- Núñez A, Vázquez-Poletti JL, Caminero AC, Castañé GG, Carretero J, Llorente IM. iCanCloud: A flexible and scalable cloud infrastructure simulator. *J Grid Comput*. 2012;10:185-209.
- Junior TJT, Bruschi S. EPCSAC–Extensible platform for cloud scheduling algorithm comparison. *Proceedings of the Anais Estendidos do XXI Simpósio em Sistemas Computacionais de Alto Desempenho*. SBC; 2020:46-53.
- Teixeira Sá T, Calheiros RN, Gomes DG. *CloudReports: An Extensible Simulation Tool for Energy-Aware Cloud Computing Environments*. Springer International Publishing; 2014:127-142.
- yaml.org. YAML Ain't Markup Language™. 2023. <https://yaml.org>
- Mastenbroek F, Andreadis G, Jounaid S, et al. OpenDC 2.0: Convenient modeling and simulation of emerging technologies in cloud datacenters. *Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE; 2021:455-464.
- Kliazovich D, Bouvry P, Khan SU. GreenCloud: A packet-level simulator of energy-aware cloud computing data centers. *J Supercomput*. 2012;62:1263-1283.
- Son J, Dastjerdi AV, Calheiros RN, Ji X, Yoon Y, Buyya R. CloudSimSDN: Modeling and simulation of software-defined cloud data centers. *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE; 2015:475-484.
- Issariyakul T, Hossain E, Issariyakul T, Hossain E. *Introduction to Network Simulator 2 (NS2)*. Springer; 2009.
- Silva Filho MC, Oliveira RL, Monteiro CC, Inácio PRM, Freire MM. CloudSim Plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness. *Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE; 2017:400-406.
- Mahmud R, Pallewatta S, Goudarzi M, Buyya R. iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments. *J Syst Softw*. 2022;190:111351.
- Mampage A, Buyya R. CloudSimSC: A toolkit for modeling and simulation of serverless computing environments. *Proceedings of the 25th IEEE International Conferences on High Performance Computing and Communications (HPCC)*; IEEE; 2023.
- Nguyen HT, Usman M, Buyya R. iQuantum: A case for modeling and simulation of quantum computing environments. *Proceedings of the 2nd IEEE International Conference on Quantum Software (QSW)*. IEEE; 2023.
- Chen W, Deelman E. WorkflowSim: A toolkit for simulating scientific workflows in distributed environments. *Proceedings of 2012 IEEE 8th International Conference on E-Science*. IEEE; 2012:1-8.
- Louis B, Mitra K, Saguna S, Åhlund C. CloudSimDisk: Energy-aware storage simulation in CloudSim. *Proceedings of 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. IEEE; 2015:11-15.
- Garg SK, Buyya R. NetworkCloudSim: Modelling parallel applications in cloud simulations. *Proceedings of 2011 Fourth IEEE International Conference on Utility and Cloud Computing*. IEEE; 2011:105-113.

28. Fittkau F, Frey S, Hasselbring W. CDOSim: Simulating cloud deployment options for software migration support. *Proceedings of 2012 IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*. IEEE; 2012:37-46.
29. Grozev N. CloudSimEx: A set of extensions for the CloudSim simulator. 2023. <https://github.com/Cloudslab/CloudSimEx>
30. Kathiravelu P, Veiga L. Concurrent and distributed CloudSim simulations. *Proceedings of 2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*. IEEE; 2014:490-493.
31. Lei WC, Chang YP, Chou LD. MINIWAN: A new framework for simulating multi-segment network topology based on mininet. *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE; 2022:105-107.
32. spec.openapis.org. OpenAPI Specification v3.1.0. 2023. <https://spec.openapis.org/oas/v3.1.0>
33. Romero-Álvarez J, Alvarado-Valiente J, Moguel E, García-Alonso J, Murillo JM. Using open API for the development of hybrid classical-quantum services. In: Troya J, Mirandola R, Navarro E, et al., eds. *Proceedings of the Service-Oriented Computing – ICSSOC 2022 Workshops*. Springer; 2023:364-368.
34. Campos M. CloudSim Plus Automation: Human-Readable Scenario Specification Tool for Automated Creation of Simulations on CloudSim and CloudSim Plus. 2023. <https://github.com/cloudsimplus/cloudsimplus-automation>
35. JaCoCo–Coverage Counter. 2023. <https://www.eclemma.org/jacoco/trunk/doc/counters.html>
36. Elahi B, Malik AW, Rahman AU, Khan MA. Toward scalable cloud data center simulation using high-level architecture. *Softw Pract Exp*. 2020;50:827-843.
37. Su X, Qi J, Wang J, Wang R, Yao Y. EasiEI: A simulator to flexibly modeling complex edge computing environments. *IEEE Internet Things J*. 2023.
38. Souza PS, Ferreto T, Calheiros RN. EdgeSimPy: Python-based modeling and simulation of edge computing resource management policies. *Futur Gener Comput Syst*. 2023;148:446-459.

**How to cite this article:** Hewage TB, Ilager S, Rodriguez MA, Buyya R. CloudSim express: A novel framework for rapid low code simulation of cloud computing environments. *Softw: Pract Exper*. 2024;54(3):483-500. doi: 10.1002/spe.3290