Future Generation Computer Systems 148 (2023) 446–459

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

EdgeSimPy: Python-based modeling and simulation of edge computing resource management policies

Paulo S. Souza [a,*], Tiago Ferreto [a], Rodrigo N. Calheiros [b]

[a] School of Technology, Pontifical Catholic University of Rio Grande do Sul, Brazil
[b] School of Computer, Data and Mathematical Sciences, Western Sydney University, Australia

ARTICLE INFO

Article history:
Received 13 January 2023
Received in revised form 17 May 2023
Accepted 13 June 2023
Available online 17 June 2023

Keywords:
Simulation
Modeling
Edge computing
Resource management
Containers
Python

ABSTRACT

The increasing popularity of applications with tight latency requirements has motivated research on Edge Computing, which positions computing resources near data sources at the Internet's edge. Despite the emergence of simulation tools that make prototype validation less complex, time-consuming, and expensive, researchers and practitioners still face significant challenges when developing resource management strategies for the edge, as existing simulators fall short in providing a fine-grained model of edge applications provisioning. To overcome this challenge, we propose EdgeSimPy, a simulation framework written in Python for modeling and evaluating resource management policies in Edge Computing environments. EdgeSimPy features a modular architecture that incorporates several functional abstractions for edge servers, network devices, and applications with built-in models for user mobility, application composition, and power consumption that allow the simulation of various scenarios. Furthermore, we propose a novel conceptual model that accurately represents the entire lifecycle of edge applications and ensures seamless integration with real application traces. In addition to submitting EdgeSimPy to an in-depth verification that checks the simulator implementation, we discuss case studies that show EdgeSimPy in action in different large-scale scenarios.

© 2023 Elsevier B.V. All rights reserved.

1. Introduction

The emerging use cases of compute-intensive applications with tight latency and bandwidth requirements have highlighted the limitations of consolidated cloud data centers [1]. In response, the IT industry has been moving towards Edge Computing [2], which distributes computing resources near data sources, enabling processing at the network's edge to avoid the long round-trip times generated by traversing the Internet backhaul [3].

While proximity to data sources grants Edge Computing a natural advantage over the cloud, it also introduces significant technical constraints. As the deployment of a large-scale edge data center in the middle of urban centers is typically unfeasible, edge infrastructures often comprise groups of devices with reduced computing power distributed across small physical spaces with limited power and cooling supply [4]. As such, ensuring the efficient use of resources, which is more critical than ever, also becomes challenging. Thus, efficient resource management policies need to be developed and tested to ensure they achieve expected goals.

In addition to the cost and time required to validate resource management policy prototypes, the distributed nature of edge infrastructures adds extra barriers to empirical experimentation of new resource management policies in real testbeds. Examples of such barriers are network instability and power outages. Such challenges favored the rise of several edge simulators, detailed in Section 3.1, that promise to close the gap between conceptual research and prototyping. Despite such initiatives, researchers and practitioners still face barriers when designing resource management policies for the edge as existing simulators fall short in providing fine-grained modeling of the edge infrastructure management, which comprises a variety of processes such as application provisioning, network flow scheduling, server maintenance, and others.

To overcome this challenge, this paper introduces EdgeSimPy, a framework for modeling and simulating resource management policies for Edge Computing environments developed in Python. EdgeSimPy is built on top of a modular architecture comprising several functional abstractions for edge servers, network devices, and applications. EdgeSimPy embodies a novel conceptual model that replicates the application provisioning method of widely

* Corresponding author.
E-mail addresses: paulo.severo@edu.pucrs.br (P.S. Souza), tiago.ferreto@pucrs.br (T. Ferreto), R.Calheiros@westernsydney.edu.au (R.N. Calheiros).

https://doi.org/10.1016/j.future.2023.06.013
0167-739X/© 2023 Elsevier B.V. All rights reserved.

used platforms such as Docker,[1] allowing seamless integration with repositories like DockerHub.[2] In addition to demonstrating the effectiveness of EdgeSimPy through an in-depth verification that checks the correctness of the simulator implementation, we describe two case studies available in the literature [5,6] that show EdgeSimPy in action in different large-scale scenarios.

The contributions of this paper are the following:

- We provide a high-level interface that leverages the features of well-known modeling solutions such as Mesa [7] and NetworkX [8] to ease the development of resource management policies and simulator extensions (Section 4.1).
- We propose a novel conceptual model that accurately represents the entire lifecycle of edge applications by replicating the behavior of widely used platforms like Docker (Section 4.3).
- We implement multiple system models for simulating several features of edge environments, such as infrastructure power consumption, application composition, service workload variations, and user mobility (Section 4.2–4.4).
- We conduct an in-depth verification that checks the correctness of EdgeSimPy's core features implementation by comparing the simulation results to the expected outputs (Section 5).
- We demonstrate EdgeSimPy's utility through two case studies based on peer-reviewed papers [5,6] that have employed the simulator in different resource management scenarios (Section 6).

The remainder of this paper is organized as follows. Section 2 reviews the concepts that shape the foundation of our research, such as Cloud Computing, Edge Computing, and containers. Then, Section 3 presents an overview of the edge simulation landscape, highlighting the contributions of EdgeSimPy over existing simulators. Next, Section 4 details EdgeSimPy's architecture and its main components. Sections 5 and 6 check EdgeSimPy's implementation correctness through an in-depth verification and demonstrate its extensibility through two case studies. Section 7, discusses lessons learned during EdgeSimPy's development, highlighting its limitations and improvement opportunities. Finally, Section 8 concludes the paper.

2. Background

Over the last decade, Cloud Computing has become one of the most attractive options for hosting applications over the Internet [9]. Cloud computing relies on centralized computing resources in data centers, reducing infrastructure management's burden and allowing providers to offer cloud services at affordable prices for individuals and organizations.

With Cloud Computing acting as a catalyst for digital transformation, the spotlight turned to use cases such as augmented reality [10] and online gaming [11], which raised the bar regarding the use of software applications in people's daily lives. Eventually, it became evident that centralization around a few cloud data centers results in them being geographically distant from end devices, resulting in response times that conflicted with the real-time requirements of those applications [12].

This limitation paved the way for the emergence of Edge Computing [2], which adopts a decentralized model where applications are hosted on resources (e.g., small servers and single-board computers) at the edge of the network, which results in shorter response times. Fig. 1 illustrates the relationship between cloud data centers, edge resources, and end devices.

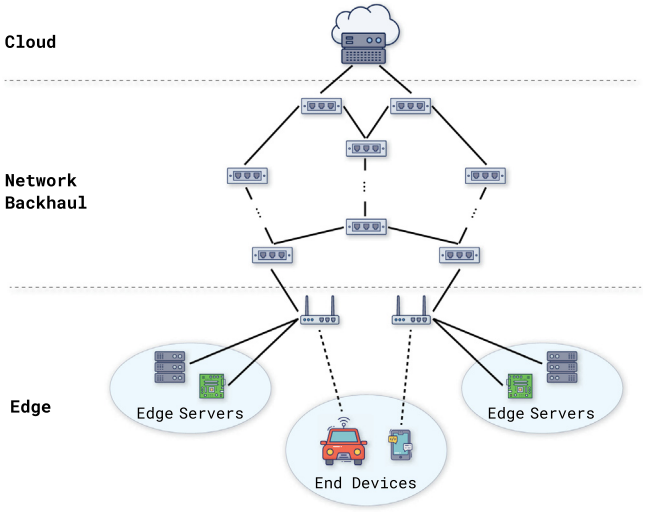


Fig. 1. Interplay between cloud data centers, edge sites, and end devices.

Similarly to cloud data centers, edge infrastructures rely on virtualization technology, often through Virtual Machines (VMs) or containers [13]. The fundamental idea behind virtualization is decoupling hardware (physical devices) from software (applications). Virtualization gives infrastructure operators fine-grained control over physical resources, which creates several resource management possibilities, such as multitenancy (i.e., hosting multiple applications on the same server) and on-the-fly relocation of applications among servers.

In addition to yielding a better use of resources, virtualization allows developers to bypass repetitive tasks during application deployment with template images [14]. At this point, the differences between VMs and containers show up. Whereas VM images are monolithic, most modern container solutions follow Docker's[3] lead, splitting container images into read-only layers representing software instructions. This difference may seem subtle, but it affects the entire lifecycle management of applications (i.e., building, deploying, and terminating operations) [15].

When a container needs to be spawned, a new writable layer is created on top of the container image's filesystem. All changes made to the container are stored in that top writable layer, so that image layers remain unchanged regardless of application-level changes, which allows containers to share common container layers [16]. On the other hand, spawning a VM requires the creation of a complete copy of the base image, as no persistence measure is taken to prevent applications from changing instructions from their base images. Consequently, provisioning a VM-based application is generally slower and generates higher disk demand than a container-based application [17].

In addition to the differences in template images, VMs and containers rely on different types of virtualization. In the VM model, a control layer, known as a hypervisor, virtualizes the hardware, giving each VM a dedicated virtual CPU, memory, I/O, and network devices. In the container model, a control layer, known as container runtime, virtualizes the host OS kernel instead of the hardware, so containers share the host OS kernel while isolation is handled at the OS level. While containerized applications do not need a standalone OS, containers display a lower virtualization overhead than VMs at the cost of weaker isolation among co-hosted instances [13].

There is no consensus about what type of virtualization should be employed on edge infrastructures. Some researchers argue

[1] https://www.docker.com/
[2] https://hub.docker.com/
[3] https://docs.docker.com/storage/storagedriver/#images-and-layers

that VMs and containers should be employed depending on the deployment needs, enabling a broader spectrum of demands to be managed more efficiently [18]. In such a line of reasoning, containers can fit better when shorter provisioning times and small footprints are needed, while VMs can deliver better security and isolation in multi-tenant deployments.

Despite the potential use cases for both types of virtualization, containers have been taking the lead as the prime architecture for deploying applications on edge computing devices, as their small footprint and low virtualization overhead fit greatly to the resource constraints of edge infrastructures while also meeting strict provisioning time constraints of edge applications [19].

At a glance, the virtualization of the edge infrastructure implies the reuse of consolidated resource management techniques already adopted in Cloud Computing. However, for the performance expectations of latency-sensitive applications running at the edge to be met, new properties such as location awareness are required to be considered, highlighting the need for new resource management approaches.

Although Edge Computing displays significant growth potential, conducting experimental research is challenging for some reasons. First, the proximity between computing resources and end devices imposes challenges related to positioning the infrastructure since deploying large-scale data centers in the middle of urban centers might not be feasible. Consequently, edge infrastructures are composed of small interconnected servers dispersed in the environment, making it difficult to conduct reproducible experiments, as multiple external factors can affect results (e.g., network instability and power supply outages).

In addition, experimental research may comprise several testing stages until prototypes are turned into consumer-ready products. Accordingly, conducting the entire experimentation process in real testbeds implies investing time and budget in computing and networking resources, power supply, and cooling with no guarantees of returns. As such, simulation tools emerge as catalysts for early research on Edge Computing, as evaluation in real testbeds is better suited for prototypes in more advanced stages.

3. Related work

Over the past decade, simulation tools like CloudSim [20] and GreenCloud [21] were broadly adopted to accelerate the development and validation of resource management strategies for cloud data centers. Similarly, the dawn of Edge Computing has motivated the development of several edge simulators. This section starts with an overview of simulation tools for Edge Computing (Section 3.1). Then, it highlights the differences and contributions of EdgeSimPy over those existing solutions (Section 3.2).

3.1. Edge computing simulators

Sonmez et al. [22] highlight the limitations of cloud and network simulators when modeling the characteristics of edge environments. While cloud simulators such as CloudSim lack user mobility and wireless support, network simulators are not focused on modeling edge servers and users. Accordingly, the authors present EdgeCloudSim, a simulator that implements user mobility, edge device power modeling, and network management, facilitating the prototyping of placement strategies in Edge Computing environments.

Qayyum et al. [23] argue that edge simulators abstract network infrastructure characteristics, restricting the options for designing new resource management strategies. The authors introduce a new simulator, FogNetSim++, which models the power consumption of edge devices, supports various communication protocols (e.g., MQTT and CoAP), and simulates the handover of mobile users among network devices. Finally, the authors present a use case demonstrating FogNetSim++ effectiveness in modeling placement and scheduling policies for the edge.

Puliafito et al. [24] discuss the critical role of application migration in preserving low latency for users despite their mobility and how edge simulators lack the features needed to evaluate migration decisions. Based on these observations, the authors propose MobFogSim, a simulator that models the application migration process based on user mobility (including attributes such as speed and moving direction) and the coverage area of access points spread in the environment.

Amarasinghe et al. [25] present a novel simulator called ECSNeT++, focused on deploying and processing stream-based applications in Edge Computing environments. Unlike other simulators, ECSNeT++ enables fine-grained resource management for streaming applications, where allocation policies can determine how tasks are processed at the core level on edge devices (where each CPU core has a processing queue), aiming to reduce the network delay and power consumption of edge devices.

Lera et al. [26] introduce YAFS, an edge simulator focused on evaluating allocation decisions for composite applications (i.e., applications composed of multiple components) on edge infrastructures. According to the YAFS model, routing policies coordinate the application communication across the network, allowing modules of a given application to be allocated on different edge devices. The authors present several YAFS use cases, including scenarios with dynamic scheduling of application components, infrastructure failures, and user mobility.

Jha et al. [27] discuss the complexity of the Cloud–Edge–IoT ecosystem, which involves allocating resources across heterogeneous devices using multiple network protocols (e.g., LoRa and Zigbee) and messaging protocols (e.g., CoAP and MQTT). After highlighting the lack of edge simulators considering the protocols of the Cloud–Edge–IoT ecosystem, the authors present the IoTSim-Edge simulator. The simulator allows the prototyping of resource management strategies for the edge based on the energy consumption of edge devices, application composition, user mobility, and communication protocols.

Alwasel et al. [28] make a case for Osmotic Computing, a paradigm that involves workload migration among cloud data centers and edge devices based on performance and security events. The authors present a novel simulator, IoTSim-Osmosis, focused on Osmotic Computing scenarios, which implements a variety of models for data transmission, energy consumption, and application performance. Finally, the authors present a case study using IoTSim-Osmosis to model policies that optimize performance, energy consumption, and cost in Cloud–Edge scenarios.

Mahmud et al. [29] highlight the lack of support for real datasets on edge simulators, which limits the evaluation of resource management policies in edge environments comprising composite applications. Then, the authors introduce iFogSim2, a simulator with native support to real datasets that incorporates modeling and simulation of service migration in multi-tier infrastructures (e.g., Cloud–Edge), user mobility, service orchestration, and edge devices clustering. They present use cases demonstrating iFogSim2 effectiveness in simulating several resource allocation scenarios at the edge.

3.2. Discussion

As discussed in the previous section, the popularization of Edge Computing motivated the development of several simulation tools that address aspects such as user mobility and infrastructure heterogeneity, which are not covered by cloud simulators despite being critical in edge scenarios.

Despite the research interest in simulation modeling focused on Edge Computing environments, existing edge simulators lack

Table 1
Summary of built-in features supported by existing simulators and EdgeSimPy.

Simulator	Server power	Network power	Network routing	User mobility	Application composition	Container lifecycle
EdgeCloudSim [22]	✘	✘	✘	✔	✘	✘
FogNetSim++ [23]	✔	✔	✔	✔	✘	✘
MobFogSim [24]	✔	✘	✘	✔	✘	✘
ECSNeT++ [25]	✔	✔	✔	✔	✔	✘
YAFS [26]	✔	✘	✔	✔	✔	✘
IoTSim-Edge [27]	✔	✘	✔	✔	✔	✘
IoTSim-Osmosis [28]	✔	✘	✔	✘	✔	✘
iFogSim2 [29]	✔	✘	✔	✔	✔	✘
EdgeSimPy (this work)	✔	✔	✔	✔	✔	✔

Table 2
Summary of potential use cases supported by existing simulators and EdgeSimPy.

Simulator	Service migration	Network flow scheduling	Container registry management	Maintenance operations
EdgeCloudSim [22]	✘	✘	✘	✘
FogNetSim++ [23]	✘	✔	✘	✘
MobFogSim [24]	✔	✘	✘	✘
ECSNeT++ [25]	✘	✔	✘	✘
YAFS [26]	✘	✘	✘	✘
IoTSim-Edge [27]	✘	✔	✘	✘
IoTSim-Osmosis [28]	✘	✔	✘	✘
iFogSim2 [29]	✔	✔	✘	✘
EdgeSimPy (this work)	✔	✔	✔	✔

native support for managing the lifecycle of containerized applications. At first, extending existing simulators may seem trivial as most of them are built with programming stacks that facilitate the inclusion of new features. However, in this case, several changes may be needed to allow the evaluation of container management strategies at the edge, such as including several new entities (e.g., container registries, images, layers, etc.) and modeling the container provisioning process from scratch, as it differs significantly from the VM model, as discussed in Section 2.

EdgeSimPy models aspects of the edge that are also supported by existing simulators, such as user mobility, energy consumption (for computing and network devices), and application composition. Complementarily, it provides functional abstractions that are not covered by existing edge simulators (e.g., container registries, container images, and container layers), enabling the simulation of the whole lifecycle of containerized applications (e.g., placement, scheduling, migration, update, and removal of operations). Table 1 summarizes the main differences between EdgeSimPy and related simulators regarding built-in features.

In addition to its built-in functionalities, EdgeSimPy aims to contribute to the community by providing more comprehensive support for specific use cases, as shown in Table 2. Although specific simulators such as MobFogSim and iFogSim2 support the modeling of service migration strategies, they focus predominantly on VM-based migration models. For instance, although MobFogSim provides a ContainerVM class, such an entity follows the VM migration model, where containers are relocated from a source to a destination server, overlooking the shareable structure of container images and the abstraction of container registries. In contrast, EdgeSimPy's abstractions for container-related entities (i.e., container registries, container images, and container layers) allow it to support the design of relocation strategies for containerized applications. In the same way, these new abstractions implemented by EdgeSimPy make room for other use cases. For instance, as EdgeSimPy represents container registries as domain-specific services, it supports the design of provisioning policies that manage such entities aiming at optimizations in the lifecycle of containerized instances. In addition, EdgeSimPy facilitates modeling maintenance operations for physical resources (e.g., servers, network devices) and applications, which is not supported by existing simulators. An in-depth analysis of EdgeSimPy's suitability in two case studies and a comparison with existing simulators in these scenarios is provided in Section 6.

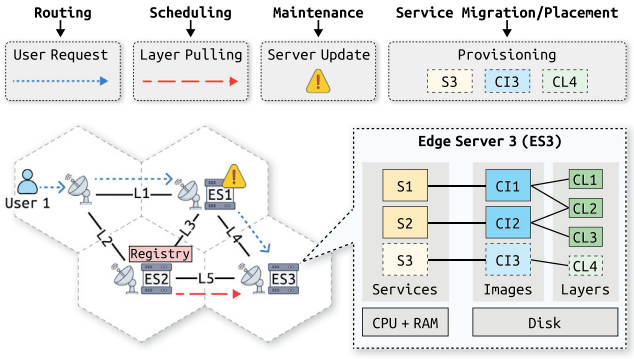


Fig. 2. Sample EdgeSimPy simulation scenario. While "L" and "ES" denote the network links and edge servers, "S", "CI", and "CL" represent the services, container images, and container layers.

4. EdgeSimPy architecture

EdgeSimPy's primary design goal is to support researchers interested in evaluating resource management strategies in Edge Computing infrastructures. For example, EdgeSimPy supports modeling different types of resource allocation decisions, such as placement, migration, scheduling, and maintenance, while considering the heterogeneity of the infrastructure (power consumption, resource capacity), users (mobility, access profile), and applications (composition, performance requirements). Fig. 2 illustrates a sample EdgeSimPy simulation scenario in which entities interact with each other and different resource management decisions affect the environment.

Before starting the simulation, EdgeSimPy expects an input file defining the simulated scenario. EdgeSimPy input files are written in the JavaScript Object Notation (JSON) format, which has a human-friendly file structure [30] and is widely adopted in various software domains [31], which facilitates EdgeSimPy integration with other tools.

EdgeSimPy input files organize the metadata of each simulated entity into a well-defined structure comprised of two distinct information groups: attributes and relationships. Attributes refer to the internal characteristics of entities, such as edge server

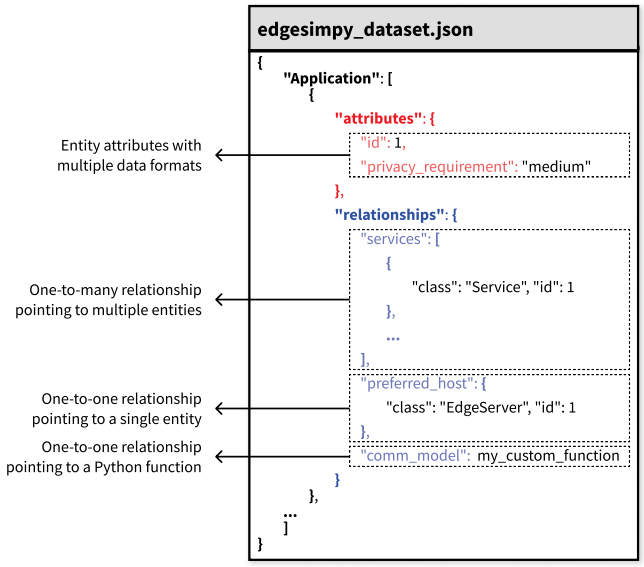


Fig. 3. Sample EdgeSimPy input file.

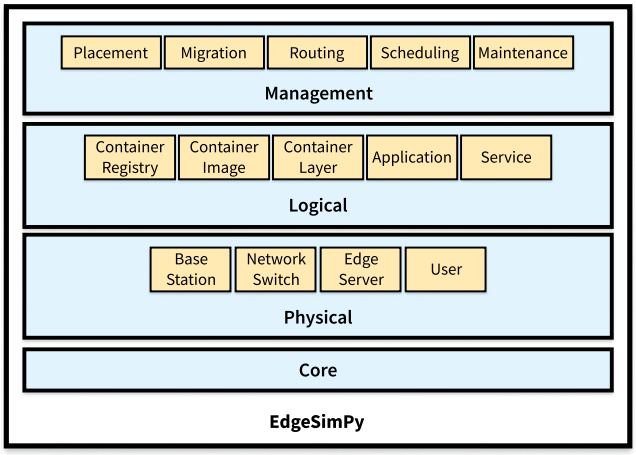


Fig. 4. EdgeSimPy architecture.

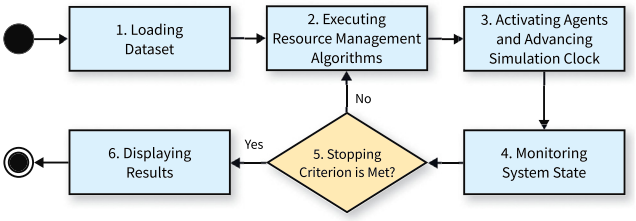


Fig. 5. EdgeSimPy's simulation workflow.

capacity, network link bandwidth, application delay, among others. Relationships represent the associations between entities (e.g., a service's host or a user's applications). By adhering to this predefined structure, EdgeSimPy can automatically identify entity input metadata and construct the simulated scenario, even in cases where custom attributes and relationships have been specified. Fig. 3 depicts a sample EdgeSimPy input file containing the metadata of a given application entity.

Simulated entities can carry geospatial metadata, which facilitates the integration of datasets containing real or synthetic coordinates to EdgeSimPy and allows the modeling of events such as user mobility. By default, EdgeSimPy uses the map model proposed by Aral et al. [32], which divides the environment into hexagonal cells.

Once the simulation starts, EdgeSimPy triggers a monitoring mechanism that stores snapshots of the entity's state at the end of each time step. Simulation logs are stored in MessagePack,[4] a binary serialization format designed to be a faster and smaller alternative to JSON [33]. Instead of writing data to disk each time step, EdgeSimPy stores the simulation output at configurable intervals of time steps, which enables reductions in the I/O pressure during the simulation.

EdgeSimPy's flexibility stems from a modular architecture (depicted as a block diagram in Fig. 4), which divides the functional abstractions into four layers (Core, Physical, Logical, and Management). Each abstraction is self-contained to streamline the integration of new features and algorithms.

The remaining of this section details the components that comprise each layer of EdgeSimPy's architecture.

4.1. Core layer

Edge Computing environments can be seen as complex systems with several entities that interact with each other in a non-linear way. This ecosystem may assemble emergent phenomena [34], which are the product of interactions between entities. An example of an emerging phenomenon at the edge is infrastructure saturation caused by a series of poor resource allocation decisions. While detecting such type of emerging phenomena is key to avoiding erroneous allocations, understanding its causes through the attributes of individual entities is challenging as it arises from the interaction between multiple entities.

One of the most popular techniques for studying emergent phenomena is Agent-Based Modeling (ABM) [35,36]. ABM-based simulations represent the world through a bottom-up approach, where independent entities (called agents) interact with each other and the environment over time according to communication and decision-making rules. Most ABM-based simulation systems adopt the Fixed-Increment Time Advance (FITA) strategy [37], which advances the simulation clock in fixed increments of time delta (Δ). If multiple events are scheduled for the same time step t, they are considered to have occurred concurrently at the end of t.

FITA-based simulators must define two properties: the granularity of Δ and the agents' activation regime [37]. Δ's granularity affects the accuracy of the simulation output and the simulation time—lower Δ yields higher accuracy to the simulator as it reduces the number of events computed in the same time step at the cost of longer runtime. The activation regime determines the order in which the simulator computes simultaneous events at the end of each time step, which can affect the simulation output, especially if simultaneous events influence each other.

EdgeSimPy employs the ABM approach, representing entities of edge scenarios as interactive agents. Features that comprise the simulation core (i.e., agent modeling, activation regime, and time advance) are managed by Mesa [7], a well-known ABM framework that ships several built-in modules that encompass the building, analysis, and visualization of ABM simulations. EdgeSimPy's simulation workflow is depicted in Fig. 5.

When EdgeSimPy is started, it loads input data from JSON files or Python dictionaries, spawning simulated entities accordingly (Fig. 5, step 1). After loading the scenario, EdgeSimPy starts an iterative process until a user-defined stopping criterion is met. At each time step, EdgeSimPy executes user-defined resource

[4] https://msgpack.org/index.html

management policies, calls the activation regime for updating the agents' state, increments the simulation clock, and collects logs about the system state, respectively (Fig. 5, steps 2–4).

Once the stopping criterion is met, EdgeSimPy stops the simulation and displays the metrics and logs collected during the simulation (Fig. 5, step 6). Thanks to EdgeSimPy's decoupled architecture, it is possible to define custom stopping criteria, resource management algorithms, and personalized routines for collecting and exhibiting simulation metrics.

4.2. Physical layer

The Physical layer contains functional abstractions for users and resources that comprise the edge infrastructure. Regardless of their distinct functions, all components in the Physical layer have a *coordinates* attribute, which carries the component's geospatial information. Physical entities that provide networking capabilities leverage the features of NetworkX [8], a well-known graph library for manipulating complex networks that ships several built-in methods (e.g., shortest path and community finding).

The remainder of this section discusses the particularities of each component in the Physical layer.

4.2.1. Base stations

Base Stations act as gateways in the edge network, providing wireless connectivity for seamless communication between users and edge servers. EdgeSimPy assumes that the base stations cover the entire map area so that users always have connectivity regardless of location. As such, the set of coordinates of the base stations comprehends the whole available area for user transit, so users cannot be in a position that represents a different coordinate from all base stations, as they would not have network connectivity. Base stations on EdgeSimPy embody multiple customizable attributes, such as energy consumption and wireless latency, enabling various scenarios to be modeled.

In addition to providing wireless connectivity, EdgeSimPy automatically handles user handoff between base stations based on user mobility patterns. Accordingly, EdgeSimPy allows for a realistic simulation of users moving through the edge network and the associated changes in connectivity as they transition from one base station to another. In addition, base stations can be equipped with network switches for wired connectivity and network flow management and edge servers for hosting services, ensuring an adaptable model for edge environments.

EdgeSimPy can also be customized to support map models where a single base station covers multiple coordinates, and attributes such as wireless latency are affected by the distance between the user's and base station's positions. This allows the modeling of various connectivity scenarios and the analysis of their impact on the performance of edge applications.

4.2.2. Network switches

Network switches provide wired connectivity between infrastructure components (e.g., base stations and edge servers) and manage data flow in the network. These components ship multiple configurable parameters, such as chassis types and varying numbers of ports with specific delay and bandwidth properties. In EdgeSimPy, network switches are modeled as nodes in the graph representing the network topology.

Although network switches are used as network nodes by default, EdgeSimPy allows other entities to be modeled for this purpose, enabling a variety of networking scenarios to be simulated. For example, cars can act as topology nodes in vehicular networks, serving the environment as intermediate data exchange and communication entities.

EdgeSimPy assumes that user requests are protected by Quality of Service (QoS) policies so that one user's request does not negatively affect others. On the other hand, more demanding data transfers between edge servers (e.g., service migrations) are modeled as network flows, sharing the bandwidth of network links. The duration of a network flow depends on the bandwidth of the links it spans over and the resource allocation policy of the network switches. Whenever a network flow starts or ends, EdgeSimPy runs a flow scheduling algorithm to update the occupation of the involved links, redistributing the available bandwidth, if applicable.

Since a network flow can use a path with links containing different bandwidth demands, EdgeSimPy normalizes the bandwidth available to a network flow to the lowest available bandwidth between the links in its path. The Max–Min Fairness algorithm [38] is used as the default flow scheduling algorithm, dividing network bandwidth proportionally to the demand of flows. As flow scheduling logic is fully encapsulated, it is possible to define custom flow scheduling algorithms without burden.

During simulation, the network's power consumption varies according to resource usage and the technical features of network switches. By default, EdgeSimPy includes the network power models proposed by Conterato et al. [39] and Rivíriego et al. [40] However, the behavior of power models is fully encapsulated, allowing new models to be implemented without requiring changes to the simulator's base features.

4.2.3. Edge servers

Edge servers are used to host services. EdgeSimPy assumes that the edge infrastructure is virtualized so that edge servers can host multiple services simultaneously. In addition to capacity parameters such as CPU, RAM, and disk storage, edge servers can also have performance parameters like Million Instructions Per Second (MIPS), which allows for an alternative representation of server performance. It is possible to define the distribution of MIPS among applications on a server based on custom criteria, allowing the modeling of advanced phenomena such as resource contention among co-hosted applications.

Technical specifications such as hardware resources and resource usage affect the power consumption of edge servers during simulation. EdgeSimPy incorporates three built-in generic power consumption models (LinearPowerModel, QuadraticPowerModel, CubicPowerModel) [41], which assume that the power consumption of edge servers grows according to their CPU usage following linear, quadratic, and cubic functions, respectively. EdgeSimPy's power modeling enables the implementation of advanced features, such as temporarily turning off edge servers to save energy. As entity properties are fully encapsulated, EdgeSimPy also supports custom edge server power models.

As edge servers have static coordinates, they are immobile by default. Nevertheless, EdgeSimPy can be extended to assign mobility models to edge servers, allowing the representation of mobile devices with computing capabilities, such as drones or Single-Board Computers (SBCs) connected to automobiles.

4.2.4. Users

As part of the Physical layer, users have a *coordinates* attribute that defines their location on the map. Users can remain in the same position during the entire simulation or move according to mobility models. By default, EdgeSimPy incorporates two mobility models, Random and Pathway [42], which can be easily replaced by other synthetic models or real mobility traces.

Users and applications are linked by a many-to-many relationship, which allows a user to access multiple applications or even an application to be accessed jointly by multiple users. Following this design principle, each user has properties that

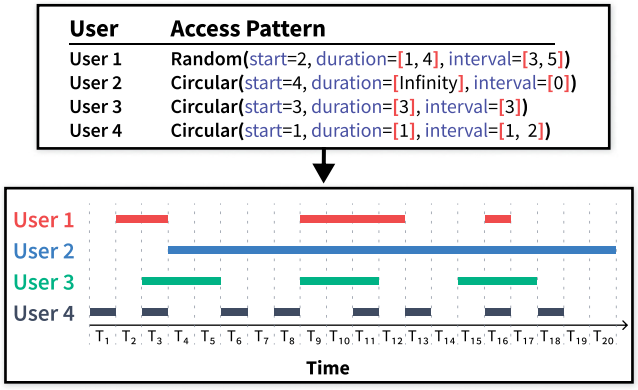


Fig. 6. EdgeSimPy's built-in user access patterns.

define their delay and availability requirements for each application they access. As each entity is self-contained, adding new user requirements such as security and budget is possible, which opens room for prototyping custom allocation strategies.

Each user has their access pattern, specifying when they will call their applications and how long each access will last. By default, EdgeSimPy incorporates two user access pattern templates, Random and Circular. While the former arbitrarily defines when and for how long the user will access their applications, the latter establishes a pattern that repeats indefinitely. Fig. 6 illustrates four users using the Random and Circular access patterns.

As the user access patterns are defined through independent classes, it is possible to easily define new patterns. This feature allows EdgeSimPy to model different workloads, from streaming to batch processing applications and serverless functions. As a user's access can be intermittent, allocation policies may choose to deprovision applications during idle periods, which can be beneficial in terms of resource saving or harmful if there are long application provisioning times after the user's request (as in the case of serverless functions facing cold starts [43]).

4.3. Logical layer

The Logical layer comprises functional abstractions for applications running on the edge infrastructure. Despite supporting VMs, EdgeSimPy adopts containerization as the default virtualization model (see details in Section 2). Accordingly, abstractions for container registries, container images, and container layers are provided. The rest of this section describes the components of the Logical layer.

4.3.1. Applications

Applications are modeled as abstract entities representing data flows involving multiple services. This way, the application services are allocated within the infrastructure rather than the applications themselves. As EdgeSimPy models applications as self-contained entities, they can receive custom attributes, such as priority and budget, which enables modeling specific scenarios.

EdgeSimPy calculates the latency of a given based on the time it takes to visit the servers that host all its services. The default application implementation in EdgeSimPy assumes that the data flow starts at the application users and passes through all the application services sequentially.

In addition to the built-in application communication behavior, EdgeSimPy supports custom communication patterns to model multiple software architectures, from monoliths to microservices [44] and stream applications [45]. It is also possible to specify custom communication policies among the services that compose a given application based on various criteria, such as inter-service latency and the characteristics of the service hosts.

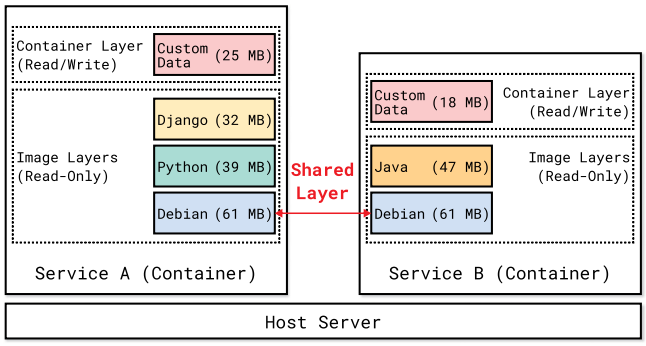


Fig. 7. Layered file system used by service in EdgeSimPy.

4.3.2. Services

Services in EdgeSimPy are modeled as container instances within the infrastructure. While a service's disk demand corresponds to the size of the layers that comprise its container image, its CPU and memory demand describe the computational resources required by the service instance and therefore are unrelated to the service's image. Each service also has a state attribute, which defines whether it is stateless or stateful.

Services leverage a layered file system model to separate service components and ensure that changes in one containerized service do not affect other co-hosted services using the same base image. This containerization approach provides an efficient and isolated environment for hosting services in the edge infrastructure. The layered file system model used by services in EdgeSimPy is shown in Fig. 7.

The layered file system used by services divides the service image into two parts: the container and image layers. In this setting, each service has its container layer with its runtime files and user session data. As the container layer is not shared with co-hosted services, it has read–write permissions. Conversely, image layers hold read-only permissions as they provide static files, libraries, and dependencies, and are shared among co-hosted services. This separation enables a streamlined relocation process and ensures that services can be efficiently transferred across edge servers without affecting other instances or interfering with shared resources.

The state attribute plays a crucial role in how service is relocated. Stateless services maintain no user session data or runtime state, allowing them to be relocated without downtime, as their container layer can be easily discarded on the target host and recreated on the destination host. In contrast, stateful services require the transfer of both image layers and the container layer containing user session data, resulting in a brief downtime while the service's state is transferred to the destination host.

4.3.3. Container registries

Container registries are the main component when allocating a service in the edge infrastructure, as the service's container image is pulled from them to the destination host. A container registry is a containerized service built on top of a registry image that embeds image distribution and storage functionality. Thus, a container registry also has its own CPU and memory requirements for performing image distribution processing.

EdgeSimPy supports the definition of custom policies for selecting from which container registries container images are pulled to the edge servers. This feature fosters the design of resource management strategies that optimize service provisioning by considering factors such as network usage and the location of container registries. Additionally, as container images follow a layered file system model, EdgeSimPy makes room for resource

management strategies that leverage multiple container registries to download the layers of a given image.

EdgeSimPy also allows defining how many layers of a particular image are downloaded simultaneously to a given host. This level of control enables the design of resource management strategies that optimize service provisioning by balancing multiple metrics of interest, such as network usage, provisioning time, and resource availability.

As container registries are modeled as domain-specific services that distribute container images across the edge infrastructure, EdgeSimPy enables dynamic provisioning of container registries as it does with regular services. This feature allows users to adapt the container registry deployment to changing network conditions, resource requirements, or application demands, ensuring efficient image distribution and optimized resource utilization in the edge environment.

4.3.4. Container images

Container images embed the basic functionality for services. By default, each image has a *tag* attribute representing its version, which enables the modeling of scenarios where new image versions are released during simulation. Like applications, container images are modeled as abstract entities, so they have no resource requirements by themselves. Instead, the disk demand of a given container image results from the size of its layers.

EdgeSimPy models container images according to the Open Container Initiative (OCI)[5] format, a well-known standard for container images. By adhering to the OCI format, EdgeSimPy can incorporate metadata from real-world container images from repositories like DockerHub, providing a more accurate representation of services in the edge infrastructure.

In addition to providing compatibility with real image traces, EdgeSimPy's container image format allows the definition of service provisioning constraints to model edge-specific scenarios. For instance, it is possible to define datasets with container image specifications that narrow the service provisioning options based on the architecture and hardware capabilities of available hosts.

4.3.5. Container layers

Container layers represent the instructions aggregated into container images. Each container layer depicts a set of files added or modified concerning the previous layer. In EdgeSimPy, container layers can be identified by a digest attribute, enabling hosts to check the integrity of downloaded container layers.

Each container layer carries attributes representing its software instruction and disk size. As container images in EdgeSimPy adhere to a layered filesystem model, co-hosted services can share read-only image data, resulting in considerable disk savings. This model makes room for the design of layer-aware resource management strategies that could minimize application provisioning time by selectively choosing hosts already possessing the necessary service layers.

4.4. Management layer

In addition to modeling the behavior of several entities of edge environments, EdgeSimPy provides fine-grained control over network and edge server resources. Consequently, it eases the prototyping of various resource management policies, such as:

- **Service Placement:** Defining the placement of application services in the infrastructure represents a vital decision to ensure the efficient use of resources. By supporting the definition of custom user access patterns, EdgeSimPy enables the modeling of online and offline placement strategies [46,47]. In offline placement scenarios, the allocation policy has *a priori* knowledge about all the applications it needs to provision. In online placement scenarios, application provisioning requests occur at runtime, and provisioning policies must allocate application services on demand.
- **Service Migration:** Given the strict latency requirements of applications running on the edge infrastructure, user mobility may require relocation of services at runtime. To support this functionality, EdgeSimPy supports the modeling of allocation policies with fine-grained control over the migration process. In this way, migration policies can define which edge servers should host the services, from which container registries the layers should be pulled, and which network paths should be used in this process.
- **Maintenance:** Updates for applications and physical devices are often released to add new features, fix bugs, or mitigate security vulnerabilities. As components of the Physical and Logical layers have versioning attributes, EdgeSimPy supports the modeling of maintenance scenarios, incorporating decisions such as the order in which components are updated.
- **Network Flow Scheduling:** In large-scale edge scenarios, events such as user mobility may trigger the provisioning of multiple applications simultaneously. However, the lack of network management can allow large flows to indiscriminately saturate the network, causing the starvation of smaller flows and reduction of the overall network performance [48,49]. Accordingly, EdgeSimPy allows the definition of scheduling strategies for network flows that can control the priority of each flow based on objectives modeled through built-in or custom attributes.

The entities comprising EdgeSimPy's architecture shape a robust platform for modeling various Edge Computing scenarios, where different resource allocation policies can be prototyped and validated without burden. In addition, each component is designed to work independently, which facilitates the inclusion of new attributes and entities to the simulator, extending the breadth of potential EdgeSimPy use cases. The next section describes the verification process used to demonstrate the correctness of EdgeSimPy implementation.

5. EdgeSimPy verification

One of the main advantages of simulation is that it allows researchers to understand and explain real-world phenomena with lower complexity and cost than empirical experimentation [50]. As modeling all the details and behaviors of a real-world system might be infeasible given the high complexity involved, simulators usually make assumptions and abstractions about the real world. While these can reduce the simulation complexity, they inherently add inaccuracies to the model [51].

One of the most critical tasks in simulation studies is checking if a simulator delivers acceptable accuracy levels given the assumptions and abstractions it implements. This task generally involves two processes: validation and verification [52]. While validation determines whether the conceptual model accurately represents the real world, verification checks whether the conceptual model's implementation is correct.

The conceptual model adopted in EdgeSimPy is based on well-known abstractions representing user mobility, network scheduling, and application provisioning, which have already been formalized and discussed in past research [42,53,54]. As EdgeSimPy adopts such abstractions without further modification, model validation is out of the scope of this work.

[5] https://opencontainers.org/

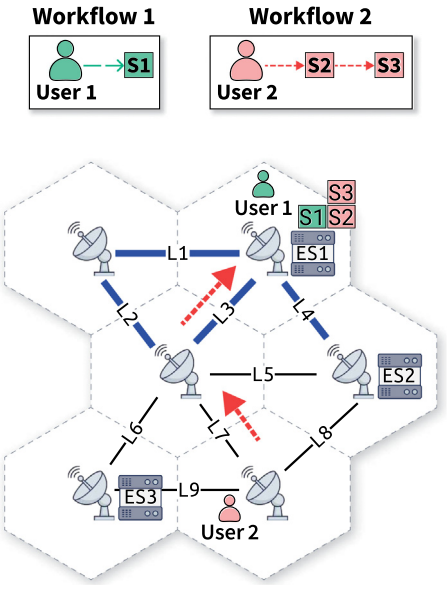


Fig. 8. Overview of the infrastructure considered in EdgeSimPy's verification. Symbols "L", "ES", and "S" denote the infrastructure's network links, edge servers, and services.

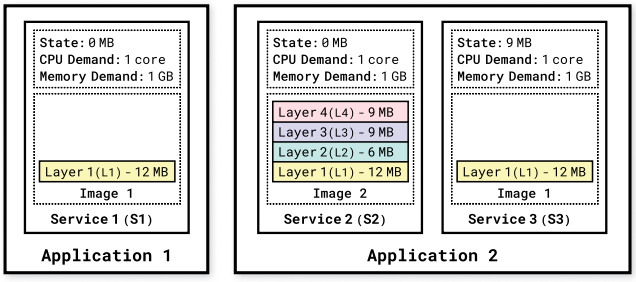


Fig. 9. Application specifications used in EdgeSimPy's verification.

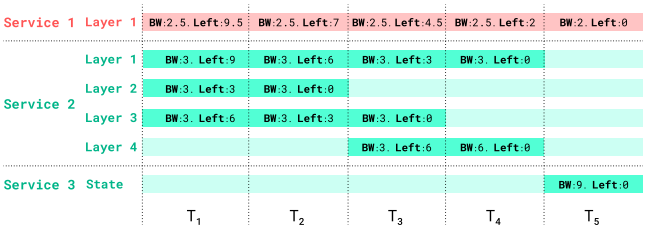


Fig. 10. Network flows used to transfer container layers and service states among servers. Here, "BW" represents available bandwidth, and "Left" is the remaining data to be transferred in future time steps.

The remaining of this section presents a verification that demonstrates the correctness of the implementation of EdgeSimPy's conceptual model. The verification process is conducted using two well-known techniques, Animation and Tracing [50], which display the simulated entities' behavior over time, allowing us to check the simulator's logic correctness.

5.1. Scenario description

Fig. 8 depicts the edge infrastructure used in EdgeSimPy's verification. We consider three NVIDIA Jetson TX2 boards representing the edge servers. Each edge server has 4 CPU cores, 8 GB of RAM, and 64 GB of storage (CPU and memory values are obtained from Süzen et al. [55]). Edge servers can download at most three container layers at once, following Docker's default configuration to avoid network congestion.[6] The edge network comprises nine links with heterogeneous capacities (for simplicity's sake, L1–L4 have a bandwidth capacity of 9 Mbps while L5–L9 have 2.5 Mbps).

In this scenario, two users move according to the Pathway model [42], each accessing one of the two existing applications. As detailed in Fig. 9, Application 1 has a stateless service (S1), while Application 2 comprises a stateless service (S2) and a stateful service (S3). Services S1 and S2 use the same base image with a single layer (L1), while S2 has a base image with four layers (L1–L4). All services are initially on edge server ES1, which also hosts a container registry demanding 1 CPU core, 1 GB of memory, and 10 MB of disk.

In the first time step, all services start to be moved out of edge server ES1. This behavior is defined through a simple resource allocation strategy to demonstrate different events within the service provisioning process. Specifically, S1 is moved to edge server ES3 through links L3 and L6, while S2 and S3 are moved to edge server ES2 through link L4. The reallocation of the three services takes six time steps. For simplicity, each time step corresponds to 1 s, and the Max–Min Fairness algorithm [53] is used as the network flow scheduling policy.

[6] https://docs.docker.com/engine/reference/commandline/pull/#concurrent-downloads

The remainder of this section discusses how EdgeSimPy events replicate the expected behavior. For such, we draw a parallel between the result of the EdgeSimPy simulation, presented in Figs. 10–11 and Table 3, and the conceptual model adopted within the simulator (i.e., containerized service provisioning and bandwidth sharing based on the Max–Min Fairness algorithm [53]).

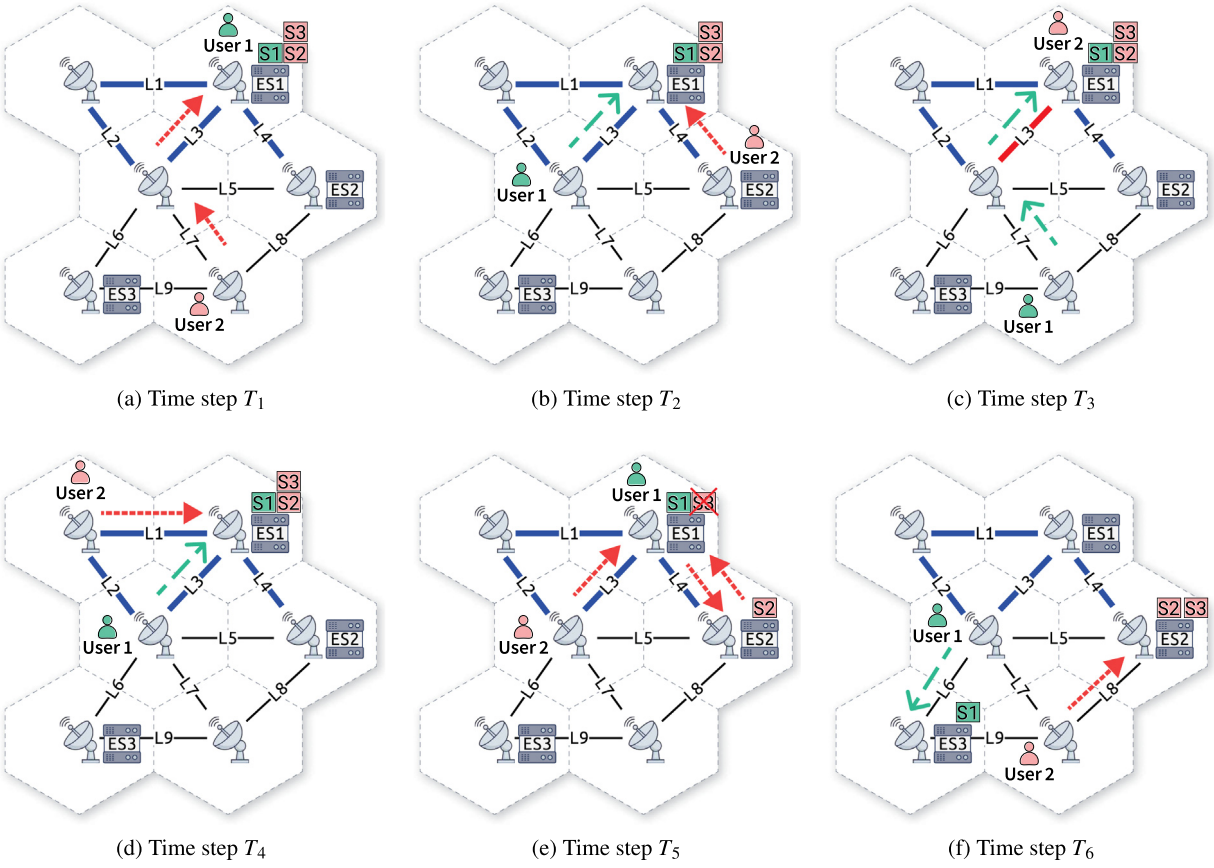
5.2. Verification discussion

Fig. 10 shows the progress of network flows spawned by service migrations. Service S1 is migrated across links L3 and L6, which have different bandwidths (9 Mbps and 2.5 Mbps, respectively). In such a scenario, EdgeSimPy equalizes the bandwidth available for service provisioning to the bandwidth of the link with the lowest capacity. Accordingly, layer L1, which corresponds to the base image of service S1, is transferred at 2.5 Mbps during time steps 1–5.

Services S2 and S3 are moved from edge server ES1 to edge server ES2 through the same path, sharing the bandwidth of link L4. As the base images of S2 and S3 are built on top of layer L1, ES2 does not pull L1 twice. Despite the layer-sharing optimization, edge server ES2 can only pull three layers at once, which forces layer L4 to wait during time steps T_1 and T_2 until the number of active downloads of ES2 decreases as the transfer of layer L2 ends.

At time step T_4, the Max–Min Fairness algorithm distributes 3 Mbps and 6 Mbps to the active flows of layers L1 and L4, respectively, instead of giving them an equal bandwidth share. That happens because Max–Min Fairness divides the available bandwidth proportionally to the size of the network flows. As layer L1's flow only needs 3 Mbps to finish, the 1.5 Mbps leftover is offered to L4's flow, which can be transferred at 6 Mbps.

The differences between stateless and stateful service provisioning are noticeable in time step T_5. Once all layers of S3's base image are present in edge server ES2, S3 is stopped on its source host ES1, and its state is transferred to ES3. EdgeSimPy keeps S3 unavailable during time step T_5 while its state is transferred, as depicted in Fig. 11. Further information about service provisioning is presented in Table 3, which details resource demand and provisioned layers on the edge servers throughout the simulation.



(a) Time step T_1 (b) Time step T_2 (c) Time step T_3

(d) Time step T_4 (e) Time step T_5 (f) Time step T_6

Fig. 11. Dynamics of each time step of EdgeSimPy's verification. Dashed arrows represent the network paths used for communication between users and services.

Table 3
Edge servers state throughout EdgeSimPy's verification simulation.

Time step	Instance	Demand			Services	Waiting queue	Download queue	Layers
		CPU	RAM	Disk				
	1	4	4096	46	Service 1, Service 2, Service 3			Registry, L1, L2, L3, L4
T_1	2	2	2048	36		L4	L1, L2, L3	
	3	1	1024	12			L1	
	1	4	4096	46	Service 1, Service 2, Service 3			Registry, L1, L2, L3, L4
T_2	2	2	2048	36		L4	L1, L3	L2
	3	1	1024	12			L1	
	1	4	4096	46	Service 1, Service 2, Service 3			Registry, L1, L2, L3, L4
T_3	2	2	2048	36			L1, L4	L2, L3
	3	1	1024	12			L1	
	1	4	4096	46	Service 1, Service 2, Service 3			Registry, L1, L2, L3, L4
T_4	2	2	2048	36				L2, L3, L1, L4
	3	1	1024	12			L1	
	1	2	2048	46	Service 1, Service 3			Registry, L1, L2, L3, L4
T_5	2	2	2048	36	Service 2			L2, L3, L1, L4
	3	1	1024	12			L1	
	1	1	1024	46				Registry, L1, L2, L3, L4
T_6	2	2	2048	36	Service 2, Service 3			L2, L3, L1, L4
	3	1	1024	12	Service 1			L1

6. EdgeSimPy case studies

This section presents two case studies from research papers that employed EdgeSimPy as a validation platform, illustrating how EdgeSimPy can be easily extended to comprehend various resource management scenarios at the edge. An in-depth discussion of each case study can be found in the original publications [5] (Section 6.1) and [6] (Section 6.2). Accordingly, this section focuses on describing the scenarios and adjustments made to EdgeSimPy for the simulations rather than discussing the results.

6.1. Case study 1: Application migration

As Edge Computing research matures, large-scale infrastructure providers are starting to move toward a new service model called Edge-as-a-Service (EaaS), which harnesses the physical proximity of the edge in an infrastructure capable of delivering the speed necessary to meet the demand of latency-sensitive applications [56,57]. Just as cloud services have led a technological shift over the past decade, EaaS offerings display great potential to become the "next big thing" in the IT industry.

This case study explored the possibility of federated edges, in which coalitions of EaaS providers are created to enhance profit and meet ever-increasing application performance expectations [58]. In such a scenario, microservice-based applications must be migrated according to user mobility while respecting privacy requirements of specific microservices.

Whereas infrastructure providers are willing to share resources to reduce application latency bottlenecks, users exhibit distinct levels of trust with different providers, which restricts allocation options for services with special privacy requirements. The performance evaluation compared a novel algorithm against three migration strategies from the literature regarding the number of migrations and SLA violations, which consider predefined latency thresholds of applications and the privacy requirements of microservices.

EdgeSimPy's base architecture does not include a functional abstraction for infrastructure providers. Therefore, to support this case study, a "*provider*" attribute is added to edge servers to identify their infrastructure providers, and a "*trusted_providers*" attribute is added to users with the providers they trust. During simulation, both parameters are checked to arrange edge servers based on the level of trust between users and infrastructure providers. In addition, a "*privacy_requirement*" attribute is added to services to specify their privacy requirement level.

The simulated scenario comprised an infrastructure with 60 edge servers managed by two infrastructure providers and 240 services with heterogeneous capacity, privacy, and latency requirements. The obtained results demonstrated that certain migration decisions, i.e., prioritizing services with special privacy requirements from applications with tight latency demands, can reduce privacy leaks by up to 7.95% at federated edges without sacrificing application latency [5].

6.2. Case study 2: Edge server maintenance

Edge infrastructures typically comprise computing resources deployed near end users, as it helps deliver low latency to applications [2]. At the same time, proximity also forces infrastructure dispersion, as deploying large-scale data centers close to urban centers may not be feasible [1]. While the distribution of computing resources allows them to be close to end users, it also introduces technical challenges related to IT operations.

In case edge resources cannot be deployed on a large scale within centralized facilities, communication between nodes typically relies on public networks, which increases the chance of instability caused by outages and lower bandwidth [59]. In addition, outdoor-deployed edge devices are more prone to physical issues and security threats. In such a scenario, infrastructure operators must invest in maintenance strategies to mitigate eventual security threats and infrastructure failures.

This case study focused on an edge server maintenance scenario. This case assumed that patches require edge servers to be rebooted for the changes to take effect. Consequently, before an edge server can be updated, the applications it hosts must be relocated to another server to avoid downtime (such a process is called server draining). As such, maintenance strategies need to schedule the server update order and define new hosts for the applications hosted by the servers that will be updated.

This case study imposed two main objectives to maintenance strategies: reduce maintenance time and application latency bottlenecks. From a security perspective, maintenance needs to be completed as soon as possible, as patches can correspond to critical security updates, so the infrastructure remains vulnerable until all edge servers are updated. At the same time, applications hosted on the infrastructure are latency sensitive, so migration decisions performed to drain servers must keep applications close enough to their users to ensure that latency remains low.

The default implementation of edge servers in EdgeSimPy comprises no attribute related to maintenance. Therefore, to support this case study, an "*updated*" attribute is added to edge servers to denote their updated status (this attribute is False by default). A "*Patch*" entity is also created with the time required to complete the update. Edge servers and patches are bounded through a relationship attribute. Finally, an "*update*()" method is implemented inside the edge server entity, representing the patching process. Once an edge server starts to be updated, it is tagged as unavailable for its patch duration, and when the update period ends, the edge server's update status is changed accordingly. In this case study, the simulation continues until all edge servers are updated.

This case study evaluation compared two novel algorithms against two strategies from the literature regarding maintenance time, the number of migrations, latency requirement violations, and vulnerability surface (which quantifies how long edge servers remain outdated during maintenance). The simulated scenario comprises an infrastructure with 40 edge servers hosting 90 applications with heterogeneous capacity and latency requirements. The results showed that user-location-aware migration decisions can reduce latency requirement violations by 30.67% on average without extending maintenance time [6].

6.3. Discussion

This section motivates the adoption of EdgeSimPy to address the needs of the described case studies, highlighting how using it is more practical than other existing simulators.

The first case study focused on migrating composite applications within an edge environment composed of multiple infrastructure providers. While many of the existing edge simulators offer support for application composition (e.g., IoTSim-Edge, YAFS, and iFogSim2), they lack functional abstractions for infrastructure providers. In addition, they do not provide features for managing the lifecycle of containers, which is the preferred virtualization technology for composite applications.

The second case study focused on edge server maintenance. Unlike EdgeSimPy, existing edge simulators do not provide the features for modeling maintenance operations, which involves incorporating version attributes to differentiate updated from outdated devices and functions to represent the update process.

Although EdgeSimPy's built-in entity attributes had to be extended for the simulations, EdgeSimPy allowed such changes to be included out of the box without requiring modifications to its base features. This was possible as EdgeSimPy automatically identifies custom entity attributes, provided its input format is followed. This feature mitigates the risk of human-induced errors arising from changes to the simulator's base features.

The ease with which EdgeSimPy allows the inclusion of custom attributes and entities is an advantage over existing simulators, which generally require creation or extension of classes to represent such attributes and entities. For example, ECSNeT++ and FogNetSim++ are built on top of OMNeT++, a discrete-event simulator written in C++. In OMNeT++, custom entities can be modeled using C++ classes called modules. Despite the flexibility of creating classes for new entities to suit specific needs, entity instances are bound by the attributes previously defined in their class definitions, meaning they cannot incorporate unique attributes not initially present in their class constructors. This constraint is also observed in EdgeCloudSim [22], MobFogSim [24], IoTSim-Edge [27], IoTSim-Osmosis [28], and iFogSim2 [29], which extend the functionality of CloudSim [20] and require extension of built-in classes or creation of new ones to represent the necessary features.

YAFS [26] displays the closest resemblance to EdgeSimPy regarding extensibility, as it is also written in Python, which is

highly flexible regarding extending classes. Nevertheless, as YAFS lacks a well-defined format for importing and exporting entities, custom components must be built manually rather than imported automatically through dataset files, as compatible serialization formats (e.g., JSON and XML) cannot recognize advanced data structures used, for example, to define relationships between objects. Additionally, manual instructions must be passed to the simulator regarding how entities are updated over the simulation time and how entity metrics should be gathered. Once importing custom entities becomes a manual process, YAFS suffers from the same issue as other simulators based on Java and C++. Whereas this behavior may not be an issue for small tests, it hinders the efficient execution of large-scale batch executions.

7. Lessons learned

Several reflections have emerged during EdgeSimPy's development, providing valuable insights for researchers and practitioners interested in developing simulation tools.

Selecting Python as the base language for EdgeSimPy has enabled users to benefit from a broad ecosystem of libraries, especially those related to emerging fields such as Artificial Intelligence. One key takeaway from our experience developing EdgeSimPy that could benefit future simulator developers involves carefully considering the base programming language, especially regarding the popularity of the language within the community and the number of available libraries that users could leverage when using the developed simulator. This approach aims to broaden the utility of the simulator, enabling users to harness community-supported algorithms and thus potentially reducing the need for manual implementation.

Throughout our internal testing, we also found that EdgeSimPy's native support for Jupyter Notebooks[7] considerably eases code sharing, as online platforms like Google Colaboratory[8] and MyBinder[9] facilitate real-time collaboration among EdgeSimPy users, eliminating the necessity for local installation of assets. With this in mind, we would advise researchers interested in simulator development to select a base programming language and design the simulator with user interactivity as a primary requirement. Leveraging interactive computing platforms can enhance the user experience through an environment that supports real-time collaboration and learning, where users can run experiments with different configurations, observe the results in real-time, and adjust their approaches accordingly.

Another lesson we can share with researchers interested in simulator development is the importance of designing a framework with a decoupled architecture. In EdgeSimPy, we have observed that its strength primarily comes from this highly decoupled structure, which includes class-based modularization and standardized input format, which facilitate the implementation of new entities and attributes, thereby supporting users with specific simulation requirements. This degree of decoupling further translates into a high level of configurability, where users can adjust simulation parameters out-of-the-box, including the simulation tick rate and system models (e.g., user mobility, power consumption of compute and networking devices, network bandwidth sharing control, etc.).

Finally, it is worth noting that appropriate calibration of the simulator parameters is critical to obtaining an accurate representation of real-world experimental setups. The calibratable parameters include the simulation tick rate and the configuration of the system model to match the scenario being modeled. For instance, this may involve defining a geographically accurate map, setting up a realistic power consumption model, or mimicking actual user mobility patterns. Additionally, the selection of metrics to be collected during the simulation is another crucial factor that must be carefully calibrated.

8. Conclusion and future work

Edge Computing is attracting attention as a paradigm that delivers low latency for applications by pulling processing from traditional cloud data centers to the network's edge [2,60]. As the expectations on the potential benefits of Edge Computing grow, ensuring that Edge Computing transitions from promise to reality require developing efficient resource management techniques.

Aside from the development challenges, experimental research at the edge can be expensive and time-consuming as it involves building and instrumenting an infrastructure with many interconnected devices. Furthermore, the distributed nature of the edge makes room for several external factors (e.g., network instability) to affect the reproducibility and reliability of results, undermining the extraction of insights needed to mature prototypes. Consequently, simulation has been considered the primary approach to accelerate the validation of early-stage prototypes at a reduced cost [61].

Existing Edge Computing simulators incorporate various features for modeling user mobility, application composition, and energy consumption of the edge infrastructure. However, they fail to deliver fine-grained control over container provisioning, which has been acknowledged as the primary choice for deploying applications at the edge.

To fill this gap, we presented EdgeSimPy, a novel simulator that models the lifecycle of containerized applications through several functional abstractions that replicate the behavior of container runtimes like Docker. In addition, EdgeSimPy features a flexible input format that allows users to define custom parameters for simulated entities out-of-the-box, extending the simulator's built-in capabilities without modifying its core features. EdgeSimPy's source code can be accessed via GitHub.[10] Furthermore, a tutorial library[11] with several practical examples has been developed to assist researchers and practitioners in effectively integrating EdgeSimPy into their work. In future work, we intend to extend EdgeSimPy networking capabilities to support the simulation of advanced routing policies that leverage the programmability of network switches at the edge.

CRediT authorship contribution statement

Paulo S. Souza: Conceptualization, Methodology, Software, Validation, Writing – original draft. **Tiago Ferreto:** Conceptualization, Supervision, Methodology, Writing – original draft, Writing – review & editing. **Rodrigo N. Calheiros:** Supervision, Methodology, Writing – original draft, Writing – review & editing.

Declaration of competing interest

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.

Data availability

No data was used for the research described in the article.

7 https://jupyter.org/

8 https://colab.research.google.com/

9 https://mybinder.org/

10 https://github.com/EdgeSimPy/EdgeSimPy

11 https://github.com/EdgeSimPy/edgesimpy-tutorials

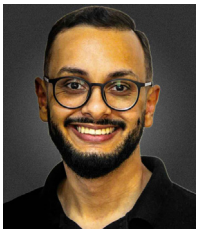
Acknowledgments

This work was supported by the PDTI Program, funded by Dell Computadores do Brasil Ltda (Law 8.248/91). The authors acknowledge the High-Performance Computing Laboratory of the Pontifical Catholic University of Rio Grande do Sul for providing resources for this project.

References

[1] M. Satyanarayanan, G. Klas, M. Silva, S. Mangiante, The seminal role of edge-native applications, in: 2019 IEEE International Conference on Edge Computing, IEEE, 2019, pp. 33–40.
[2] M. Satyanarayanan, P. Bahl, R. Caceres, N. Davies, The case for vm-based cloudlets in mobile computing, IEEE Pervasive Comput. 8 (4) (2009) 14–23.
[3] H. Zhao, S. Deng, Z. Liu, J. Yin, S. Dustdar, Distributed redundancy scheduling for microservice-based applications at the edge, IEEE Trans. Serv. Comput. (2020).
[4] J. Wang, Z. Feng, S. George, R. Iyengar, P. Pillai, M. Satyanarayanan, Towards scalable edge-native applications, in: ACM/IEEE Symposium on Edge Computing, 2019, pp. 152–165.
[5] P. Souza, A. Crestani, T. Ferreto, F. Rossi, Latency-aware privacy-preserving service migration in federated edges, in: International Conference on Cloud Computing and Services Science, 2022, pp. 288–295.
[6] P. Souza, T. Ferreto, F. Rossi, R. Calheiros, Location-aware maintenance strategies for edge computing infrastructures, IEEE Commun. Lett. 26 (4) (2022) 848–852.
[7] J. Kazil, D. Masad, A. Crooks, Utilizing python for agent-based modeling: The mesa framework, in: International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation, Springer, 2020, pp. 308–317.
[8] A. Hagberg, P. Swart, D. S. Chult, Exploring network structure, dynamics, and function using NetworkX, Technical Report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
[9] R. Buyya, S.N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L.M. Vaquero, M.A. Netto, et al., A manifesto for future generation cloud computing: Research directions for the next decade, ACM Comput. Surv. 51 (5) (2018) 1–38.
[10] R. Roman, J. Lopez, M. Mambo, Mobile edge computing, fog and others: A survey and analysis of security threats and challenges, Future Gener. Comput. Syst. 78 (2018) 680–698.
[11] S. Shahzadi, M. Iqbal, T. Dagiuklas, Z.U. Qayyum, Multi-access edge computing: Open issues, challenges and future perspectives, J. Cloud Comput. 6 (1) (2017) 1–13.
[12] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2322–2358.
[13] P. Sharma, L. Chaufournier, P. Shenoy, Y.C. Tay, Containers and virtual machines at scale: A comparative study, in: International Middleware Conference, ACM, New York, NY, USA, 2016, pp. 1–13.
[14] K. Wang, J. Rao, C.-Z. Xu, Rethink the virtual machine template, ACM SIGPLAN Not. 46 (7) (2011) 39–50.
[15] C.-P. Wu, M.A. Suresh, D. Da Silva, Container lifecycle management for edge nodes: poster, in: ACM/IEEE Symposium on Edge Computing, 2017, pp. 1–2.
[16] J. Darrous, T. Lambert, S. Ibrahim, On the importance of container image placement for service provisioning in the edge, in: International Conference on Computer Communication and Networks, IEEE, 2019, pp. 1–9.
[17] C.G. Kominos, N. Seyvet, K. Vandikas, Bare-metal, virtual machines and containers in OpenStack, in: 2017 20th Conference on Innovations in Clouds, Internet and Networks, Icin, IEEE, 2017, pp. 36–43.
[18] K. Ha, Y. Abe, T. Eiszler, Z. Chen, W. Hu, B. Amos, R. Upadhyaya, P. Pillai, M. Satyanarayanan, You can teach elephants to dance: Agile VM handoff for edge computing, in: ACM/IEEE Symposium on Edge Computing, 2017, pp. 1–14.
[19] B.I. Ismail, E.M. Goortani, M.B. Ab Karim, W.M. Tat, S. Setapa, J.Y. Luke, O.H. Hoe, Evaluation of docker as edge computing platform, in: IEEE Conference on Open Systems, IEEE, 2015, pp. 130–135.
[20] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A. De Rose, R. Buyya, CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw. - Pract. Exp. 41 (1) (2011) 23–50.
[21] D. Kliazovich, P. Bouvry, S.U. Khan, GreenCloud: A packet-level simulator of energy-aware cloud computing data centers, J. Supercomput. 62 (3) (2012) 1263–1283.
[22] C. Sonmez, A. Ozgovde, C. Ersoy, EdgeCloudSim: An environment for performance evaluation of edge computing systems, in: International Conference on Fog and Mobile Edge Computing, 2017, pp. 39–44.
[23] T. Qayyum, A.W. Malik, M.A. Khan Khattak, O. Khalid, S.U. Khan, FogNetSim++: A toolkit for modeling and simulation of distributed fog environment, IEEE Access 6 (2018) 63570–63583.
[24] C. Puliafito, D.M. Gonçalves, M.M. Lopes, L.L. Martins, E. Madeira, E. Mingozzi, O. Rana, L.F. Bittencourt, MobFogSim: Simulation of mobility and migration for fog computing, Simul. Model. Pract. Theory 101 (2020) 102062.
[25] G. Amarasinghe, M.D. de Assunção, A. Harwood, S. Karunasekera, ECSNeT++ : A simulator for distributed stream processing on edge and cloud environments, Future Gener. Comput. Syst. 111 (2020) 401–418.
[26] I. Lera, C. Guerrero, C. Juiz, YAFS: A simulator for IoT scenarios in fog computing, IEEE Access 7 (2019) 91745–91758.
[27] D.N. Jha, K. Alwasel, A. Alshoshan, X. Huang, R.K. Naha, S.K. Battula, S. Garg, D. Puthal, P. James, A. Zomaya, S. Dustdar, R. Ranjan, IoTSim-edge: A simulation framework for modeling the behavior of Internet of Things and edge computing environments, Softw. - Pract. Exp. 50 (6) (2020) 844–867.
[28] K. Alwasel, D.N. Jha, F. Habeeb, U. Demirbaga, O. Rana, T. Baker, S. Dustdar, M. Villari, P. James, E. Solaiman, R. Ranjan, IoTSim-Osmosis: A framework for modeling and simulating IoT applications over an edge-cloud continuum, J. Syst. Archit. 116 (2021) 101956.
[29] R. Mahmud, S. Pallewatta, M. Goudarzi, R. Buyya, iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments, J. Syst. Softw. 190 (2022) 111351.
[30] N. Nurseitov, M. Paulson, R. Reynolds, C. Izurieta, Comparison of JSON and XML data interchange formats: A case study, in: International Conference on Computer Applications in Industry and Engineering, 2009, pp. 157–162.
[31] F. Pezoa, J.L. Reutter, F. Suarez, M. Ugarte, D. Vrgoč, Foundations of JSON schema, in: International Conference on World Wide Web, 2016, pp. 263–273.
[32] A. Aral, V. De Maio, I. Brandic, ARES: Reliable and sustainable edge provisioning for wireless sensor networks, IEEE Trans. Sustain. Comput. 7 (4) (2021) 761–773.
[33] J. Vanura, P. Kriz, Perfomance evaluation of Java, JavaScript and PHP serialization libraries for XML, JSON and binary formats, in: International Conference on Services Computing, Springer, 2018, pp. 166–175.
[34] H. Sayama, Introduction to the Modeling and Analysis of Complex Systems, Open SUNY Textbooks, 2015.
[35] C.M. Macal, M.J. North, Tutorial on agent-based modeling and simulation, in: Winter Simulation Conference, IEEE, 2005, pp. 14–pp.
[36] E. Bonabeau, Agent-based modeling: Methods and techniques for simulating human systems, Proc. Natl. Acad. Sci. 99 (2002) 7280–7287.
[37] A.M. Law, W.D. Kelton, W.D. Kelton, Simulation Modeling and Analysis, Vol. 5, Mcgraw-hill New York, 2015.
[38] D.P. Bertsekas, R.G. Gallager, P. Humblet, Data Networks, Vol. 2, Prentice-Hall International New Jersey, 1992.
[39] M.d.S. Conterato, T.C. Ferreto, F. Rossi, W.d.S. Marques, P.S.S. de Souza, Reducing energy consumption in SDN-based data center networks through flow consolidation strategies, in: ACM/SIGAPP Symposium on Applied Computing, 2019, pp. 1384–1391.
[40] P. Reviriego, J.A. Hernández, D. Larrabeiti, J.A. Maestro, Performance evaluation of energy efficient ethernet, IEEE Commun. Lett. 13 (9) (2009) 697–699.
[41] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, Concurr. Comput.: Pract. Exper. 24 (13) (2012) 1397–1420.
[42] F. Bai, A. Helmy, A survey of mobility models, in: Wireless Adhoc Networks, Vol. 206, University of Southern California, USA, 2004, p. 147.
[43] P. Silva, D. Fireman, T.E. Pereira, Prebaking functions to warm the serverless cold start, in: International Middleware Conference, 2020, pp. 1–13.
[44] F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro, S. Cretti, Throughput-aware partitioning and placement of applications in fog computing, IEEE Trans. Netw. Serv. Manag. 17 (4) (2020) 2436–2450.
[45] M.D. de Assuncao, A. da Silva Veith, R. Buyya, Distributed data stream processing and edge computing: A survey on resource elasticity and future directions, J. Netw. Comput. Appl. 103 (2018) 1–17.
[46] L. Zhao, L. Lu, Z. Jin, C. Yu, Online virtual machine placement for increasing cloud provider's revenue, IEEE Trans. Serv. Comput. 10 (2) (2015) 273–285.
[47] H. Tian, J. Wu, H. Shen, Efficient algorithms for VM placement in cloud data centers, in: International Conference on Parallel and Distributed Computing, Applications and Technologies, IEEE, 2017, pp. 75–80.
[48] J. Shi, O. Gurewitz, V. Mancuso, J. Camp, E.W. Knightly, Measurement and modeling of the origins of starvation in congestion controlled mesh networks, in: International Conference on Computer Communications, IEEE, 2008, pp. 1633–1641.
[49] C. Toprak, C. Turker, A.T. Erman, Detection of DHCP starvation attacks in software defined networks: A case study, in: International Conference on Computer Science and Engineering, IEEE, 2018, pp. 636–641.

[50] X. Xiang, R. Kennedy, G. Madey, S. Cabaniss, Verification and validation of agent-based scientific simulation models, in: Agent-Directed Simulation Conference, Vol. 47, The Society for Modeling and Simulation International San Diego, CA, USA, 2005, p. 55.
[51] R.G. Sargent, Verification and validation of simulation models, in: Conference on Winter Simulation, IEEE, 2010, pp. 166–183.
[52] O. Balci, Principles and techniques of simulation validation, verification, and testing, in: Conference on Winter Simulation, 1995, pp. 147–154.
[53] F. Gebali, Analysis of Computer and Communication Networks, Springer Science & Business Media, 2008.
[54] L.A.D. Knob, C.H. Kayser, P.S.S. de Souza, T. Ferreto, Enforcing deployment latency SLA in edge infrastructures through multi-objective genetic scheduler, in: IEEE/ACM International Conference on Utility and Cloud Computing, 2021, pp. 1–9.
[55] A.A. Süzen, B. Duman, B. Şen, Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn, in: International Congress on Human-Computer Interaction, Optimization and Robotic Applications, IEEE, 2020, pp. 1–5.
[56] N. Chen, Y. Yang, T. Zhang, M.-T. Zhou, X. Luo, J.K. Zao, Fog as a service technology, IEEE Commun. Mag. 56 (11) (2018) 95–101.
[57] A. Jindal, G.S. Aujla, N. Kumar, SURVIVOR: A blockchain based edge-as-a-service framework for secure energy trading in SDN-enabled vehicle-to-grid environment, Comput. Netw. 153 (2019) 36–48.
[58] C. Anglano, M. Canonico, P. Castagno, M. Guazzone, M. Sereno, A game-theoretic approach to coalition formation in fog provider federations, in: International Conference on Fog and Mobile Edge Computing, IEEE, 2018, pp. 123–130.
[59] A. Aral, I. Brandić, Learning spatiotemporal failure dependencies for resilient edge computing services, IEEE Trans. Parallel Distrib. Syst. 32 (7) (2020) 1578–1590.
[60] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39.
[61] M. Salama, Y. Elkhatib, G. Blair, IoTNetSim: A modelling and simulation platform for end-to-end IoT services and networking, in: International Conference on Utility and Cloud Computing, 2019, pp. 251–261.



Paulo S. Souza is a Ph.D. candidate in Computer Science at the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil. He received his Master's degree in Computer Science from the same institution in 2020. His research interest lies primarily in the fields of computational resource management, Cloud Computing, Edge Computing, and algorithms.

Tiago Ferreto is an Associate Professor in Computer Science at the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil, where he obtained his Ph.D. degree in Computer Science in 2010. His primary academic research interests are resource management, Cloud Computing, Edge Computing, and virtualization.

Rodrigo N. Calheiros is an Associate Professor at the School of Computer, Data and Mathematical Sciences, Western Sydney University, Australia. He works in the field of Cloud Computing and related areas since 2008, and since then he carried out R&D supporting research in the area. His research interests also include Big Data, Internet of Things, Fog Computing, and their application.