electronics

MDPI

Article

VonEdgeSim: A Framework for Simulating IoT Application in Volunteer Edge Computing

Yousef Alsenani

Infomation System Department, FCIT, King Abdulaziz University, Jeddah 21589, Saudi Arabia; yalsenani@kau.edu.sa

Abstract: Recently, various emerging technologies have been introduced to host IoT applications. Edge computing, utilizing volunteer devices, could be a feasible solution due to the significant and underutilized resources at the edge. However, cloud providers are still reluctant to offer it as an edge infrastructure service because of the unpredictable nature of volunteer resources. Volunteer edge computing introduces challenges such as reliability, trust, and availability. Testing this infrastructure is prohibitively expensive and not feasible in real-world scenarios. This emerging technology will not be fully realized until dedicated research and development efforts have substantiated its potential for running reliable services. Therefore, this paper proposes VonEdgeSim, a simulation of volunteer edge computing. To the best of our knowledge, it is the first and only simulation capable of mimicking volunteer behavior at the edge. Researchers and developers can utilize this simulation to test and develop resource management models. We conduct experiments with various IoT applications, including Augmented Reality, Infotainment, and Health Monitoring. Our results show that incorporating volunteer devices at the edge can significantly enhance system performance by reducing total task delay, and improving task execution time. This emphasizes the potential of volunteers to provide reliable services in an edge computing environment. The simulation code is publicly available for further development and testing.

Keywords: volunteer edge computing; edge computing; Internet of Things; simulation; trust management; fault tolerance

Citation: Alsenani, Y. VonEdgeSim: A Framework for Simulating IoT Application in Volunteer Edge Computing. *Electronics* **2024**, *13*, 4124. https://doi.org/10.3390/electronics13204124

Academic Editor: Carlo Mastroianni

Received: 19 July 2024
Revised: 15 October 2024
Accepted: 16 October 2024
Published: 19 October 2024

1. Introduction

Cloud computing provides centralized, powerful resources such as storage and networking, allowing businesses and individuals to access virtualized resources. However, sensitive applications might not be well suited to run on these centralized cloud data centers due to latency issues [1]. Since latency-sensitive applications perform poorly when hosted in distant data centers, new paradigms have emerged, namely edge and fog computing [2]. These paradigms introduce resources closer to the user, such as edge servers and routers, positioned between the cloud and the end-user. This allows providers to enable users to deploy and process their applications and data at nearby endpoints, making these services ideal for ultra-low latency and real-time responsiveness (https://aws.amazon.com/edge/ (accessed on 17 September 2024)).

On the other hand, dew computing or volunteer edge computing is a paradigm that leverages unused resources, such as personal computers and devices [3]. Volunteer edge computing (VEC) can be promising infrastructure for this application if they are utilized effectively. Volunteer edge computing is a concept that uses edge computing and volunteer computing to maximize unused computer resources at the edge of the network for Internet of Things applications. Volunteer devices are highly scalable and close to user applications [4,5]. However, providers still hesitate to deploy user tools and API on any nearby volunteer edge device due to the nature of the volunteer devices.

Volunteer infrastructure presents challenges related to reliability, availability, and trust [3,6–8]. Volunteer infrastructure is not like dedicated physical servers in cloud data

centers or edge servers. These resources are non-dedicated, as they are personal devices. Their nature greatly affects the overall system performance and reliability. Volunteer devices might go offline at any time, which can lead to task failures or delays. Moreover, the trust score of volunteer devices cannot be assumed. Volunteer devices register their resources for donation or participation without prior knowledge or history of trustworthiness.

Several successful volunteer computing projects, such as Boinc (https://boinc.berkeley.edu/ (accessed on 17 September 2024)), and Folding@home (https://foldingathome.org/ (accessed on 17 September 2024)) are built to utilize idle resources to run scientific applications such as medicine, biology, and astronomy. For example, Boinc consists of 50 projects established for over two decades. The success of these projects comes from different reasons, such as dedicated research conducted on volunteer computing resource management and fault tolerance. At the same time, some projects started before the revelation of the hypervisor layer that the cloud provides. However, unique challenges are introduced when it comes to volunteer edge computing, such as the quality of service (QoS), and quality of experience (QoE) [9,10]. Only a few research studies have been conducted in the direction of resource management of VEC because running and setting volunteer edge computing is not feasible. Additionally, there is no VEC simulation available to assist the research community.

Evaluating the proposed framework or hypothesis in a real system gives the best result. However, setting up a real system requires cost, maintenance, and reconfiguration, which is unavailable in some research communities and labs [11]. The nature of volunteer infrastructure brought many challenges to testing methodology on these simulators or a real system. First, scalability is the main challenge, especially when testing on a large scale of devices to take full advantage of the concept of volunteer edge. Second, testing any method or hypothesis requires running the test in multiple rounds, which is challenging if the system is dynamic [12]. Another crucial additional reason is the trust of volunteers. Running a real test without a prior trust model might be risky for other entities in the test [13]. The alternative solution is to evaluate and run the experiment on a simulation that mimics the system's complexity, dynamicity, and heterogeneity. Number cloud and edge computing simulators such as [14,15] exist to mimic the characteristics of cloud and edge environments. However, VonEdgeSim stands out by simulating failures and incorporating trust modeling to replicate the non-dedicated environment inherent in volunteer computing within edge networks.

Our research question is as follows: how can scalability, trust, and resource management issues be effectively addressed through the simulation of volunteer edge computing? The development and assessment of our suggested solution, VonEdgeSim, which attempts to close a sizable gap in the body of current research, are guided by this question. While current simulators such as CloudSim and EdgeCloudSim provide useful tools for cloud and edge computing, they are unable to mimic the main challenges of volunteer edge computing, including non-dedicated resources and dynamicity of trust resource. VonEdgeSim is the first simulation with such sophisticated features, bridging this gap by offering a thorough simulation environment that takes these important elements into account.

Therefore, in this paper, we propose VonEdgeSim, a volunteer edge computing simulation. The simulation assists the research community in testing their proposed resource management approach on simulation before integrating it into the real system. The simulation incorporates the dynamic infrastructure of volunteer edge devices alongside dedicated resources such as cloud data centers and edge servers. The task assignment considers important features such as CPU speed, memory size, bandwidth, energy consumption, trust of volunteer edge, and latency. This simulation is written in Python code, making it easy for non-experts to deploy their features and task characteristics. By offering a robust platform for simulating volunteer edge computing scenarios, VonEdgeSim enables researchers to test and refine their approaches before real-world deployment, potentially leading to more reliable and efficient edge computing solutions. To the best of our knowledge, this is the first and only simulation with the above feature, making it suitable for the research

community to evaluate the proposed work. Several tests are conducted to test the usage and capabilities of simulation.

The following is a summary of this article's contributions:

1. The related simulations in cloud, edge, and volunteer computing are discussed in detail. This section presents several well-known simulations in related paradigms such as cloud computing, edge, fog computing, volunteer computing, and desktop grids. We distinguish the different features that our simulation addresses.
2. The design and implementation of simulation is the core of this work and is presented in great detail. First, the volunteer edge computing environment is briefly presented. The simulation design is described in detail, including the network, task, simulation class, runtime control, random initialization, and the implementation.
3. The design of a benchmark case is discussed, and several experiment tests are conducted, such as delay, task failure, task execution time, and scalability.

The remainder of this paper is organized as follows. We introduce the Section 2. Section 3 discusses the related work. We briefly define the volunteer edge computing in Section 4. In Section 5, the design of the simulation is discussed in great detail. The implementation is presented in Section 6. The experiment are presented in Section 7. Finally we present the conclusion in Section 8.

2. Motivation

There are a number of simulators designed for cloud, fog, and edge computing. CloudSim [15] is a well-known toolkit that evaluates the performance of power consumption and scheduling for centralized datacenters. The virtualized layer for virtual machines is a core feature that makes CloudSim a successful simulator for assessing resource allocation mechanisms before applying them to real datacenters. Additionally, many toolkits have been built on top of and as extensions to CloudSim to simulate the resource management needed for IoT environments. The infrastructure and physical servers built on CloudSim are dedicated and static, which can be used for evaluating resource management for volunteer computing. Moving to shifted paradigms such as fog computing has gained attention to address some CloudSim limitations in simulating this shifted environment. iFogSim [16] is an extension of CloudSim to simulate fog computing. The simulator is designed to evaluate the constrained latency applications that run on fog networks. Even though the current version of iFogSim supports mobility, migration, and microservice orchestration, it lacks features that simulate volunteer device behavior. Similarly, for edge computing environments, EdgeCloudSim [14] is a popular simulator. EdgeCloudSim consists of different network communications such as WLAN, WAN, and LAN, and has many great features for edge environments.

On the other hand, volunteer edge computing can complement cloud data centers for different types of IoT applications. There have been several pieces of research addressing resource management in volunteer edge computing. However, each of these studies conducted experiments in different environments, making it difficult for the community to process and make VEC feasible for business network providers. Volunteer edge computing faces significant challenges that have hindered its adoption in business settings. Network providers are still hesitant to deploy user tools over volunteer infrastructure due to these challenges. The theoretical background and research challenges in this field that require simulation-based solutions have not been adequately addressed by the existing simulations. The main challenges of volunteer edge computing include availability, reliability, and trust. The dynamic environment allows volunteers to leave and join in a random and unpredictable manner, which has a significant impact on task performance, often resulting in failures and delays. The trust model is a crucial component that must be added to resource management to guarantee system performance.

Moreover, most of the existing simulators are written in Java or C programming languages, which limits their accessibility to users who are experts in these languages. Therefore, we introduce a novel Python-based simulator for volunteer edge computing called

VonEdgeSim. This simulator differs from existing work by addressing more challenges that help the community evaluate their proposed solutions in volunteer edge computing before applying them to real environments. The use of Python makes this simulator more accessible, as it is widely used by communities beyond computer science and engineering. Additionally, the numerous features and libraries available in Python make this simulator a strong base for extension and further development. Discussing the above challenges, capabilities, and limitations of these simulators highlights the need for designing simulators specifically for volunteer edge computing. We introduce new challenges that are core to simulating VEC.

3. Related Work

In this section, we discuss approaches that integrate volunteer computing and edge computing. Second, we discuss related simulations: (a) cloud computing simulators, (b) edge/fog and IoT simulations, and (c) volunteer computing simulations. Then, we justify our simulation approach. Finally, we identify the gaps in current simulations.

Because volunteer edge computing can interact with multiple paradigms, a thorough search across multiple areas is required. Using a variety of academic resources, such as IEEE Xplore, ACM Digital Library, and Google Scholar, we carried out this thorough literature search. In addition to "toolkits" related to these paradigms, specific search strings like "volunteer edge computing", "IoT simulation", "cloud computing simulators", "edge computing simulators", "resource management in edge computing", "Desktop Grid simulators", "Dew Computing simulators", "Fog Computing simulation" and "Mobile Computing simulators" were used. Recent papers, applicability to volunteer edge computing, and characteristics of the toolkit or simulation were among the selection factors. Works that did not explicitly address the special difficulties associated with volunteer edge computing were disregarded.

Volunteer edge computing: The integration of volunteer computing and edge computing has recently been gaining attention [5]. These are two different infrastructures. Volunteer computing, also known as desktop grid computing, has been a very successful project over the past two decades [17]. Well-known projects such as BOINC [18,19] and Folding@home [20] were built specifically for scientific applications that require intensive, large-scale CPU resources. Volunteers donate and register their resources, such as CPU power, to be utilized for these scientific applications. On the other hand, edge computing is a new paradigm developed to accommodate latency-sensitive applications that cannot be efficiently handled by cloud computing alone. These applications can run on edge servers rather than traveling longer distances to cloud data centers. Both infrastructures have been widely and successfully used.

However, there is no known project or system currently in operation that effectively integrates volunteer and edge computing to enhance the concept of edge computing, similar to how volunteer computing projects are actively running today. There are a number of earlier and recent research studies discussing resource management in volunteer edge computing (VEC).

Earlier work on volunteer edge computing: A study by Alsenani et al. [10] is one of the early works that discusses volunteer edge computing. Alsenani et al. proposed a statistical approach to estimate the reliability and trust of devices. The method is based on the Beta distribution, a function that has been widely used in trust modeling. The Beta distribution is defined by two parameters that shape the distribution. Alsenani's approach uses the success and failure of tasks as the criteria to estimate the trustworthiness of volunteer devices. Pandey et al. [21] propose VECTrust, a trusted framework for resource allocation in volunteer edge cloud (VEC) computing. VECTrust is based on performance, agility, cost, and security (PACS) factors. It uses a Dirichlet-based probability distribution to leverage multiple variables, unlike SaRa [10], which relies on only two variables. VECTrust was evaluated using bioinformatics workflows on a VEC testbed with KubeEdge and Docker technologies. The experiments outperformed state-of-the-art reliability-based trust models

and showed promising results, demonstrating that volunteer resources can be successfully utilized in edge networks. Mengistu et al. [22] propose integrating volunteer computing with edge computing. The authors propose a three-layer architecture that includes the public cloud layer, the mini data center (VCaaS layer), and the front-end edge devices/applications (user layer). The system uses vertical and/or horizontal communication between layers. The authors present several open challenges in this infrastructure, such as offloading and partitioning, communication models, business models, quality of service, and security.

Recent advances in volunteer edge computing: Mounesan et al. [23] propose a Reinforcement Learning (RL)-driven approach for scheduling data-intensive scientific workflows in volunteer edge cloud (VEC) environments. The problem is formulated using a Markov Decision Process (MDP) and solved with an Asynchronous Advantage Actor–Critic (A3C) based RL model. The work demonstrates that better utilization of volunteer resources can result in reduced task rejection compared to traditional methods. Alarcon et al. propose VECFlex [24], a flexible resource management framework for VEC. The framework consists of two key features: resource security policies and efficient resource allocation. VECFlex uses Reinforcement Learning (RL) to rank resource trust and Particle Swarm Optimization (PSO) to allocate optimal resources to workflow tasks. VECFlex was evaluated on an AWS testbed, demonstrating a 2x improvement in workflow execution latency. Kantardjian et al. [25] propose the Dynamic Worker Availability Prediction (DWAP) scheme for extreme edge computing. DWAP uses a continuous-time Markov model to predict the availability of workers. The scheme is evaluated using real Google cluster traces. DWAP outperforms state-of-the-art methods in terms of task drop rate.

Existing simulators: There has been a great shift from centralized computing paradigms, such as cloud computing, to decentralized paradigms like IoT and edge computing. There are numerous simulators available, each designed for specific purposes and infrastructures, including cloud computing, edge computing, and volunteer computing. In the following sections, we discuss each of these simulators in detail.

In cloud computing, CloudSim [15] is an event-derived solution toolkit used to model the behavior of a cloud computing data center. CloudSim consists of great features that mimic the behavior of virtualized server hosts, containers, and applications. It also supports message-passing applications in the federation cloud and is suitable for testing energy-efficient mechanisms. DCSim [26] is a data centre simulation tool for evaluating dynamic virtualized resource management. DCSim supports the multi-tier application paradigm, which enables the use of virtual machine replication as a tool for managing growing workloads. DCSim allows the ability to simulate dependencies between VMs, and the combination of these capabilities with a work-saving CPU scheduler. These simulators are designed for simulating data centers and does not support an edge computing infrastructure.

EdgeCloudSim [14] is a popular simulation edge computing system built on top of ClousSim. EdgeCloudSim supports both computational and networking resources. WLAN and WAN device mobility models are available in the simulation. A number of edge application scenarios are available to test. IoTSim-Edge [12] is an IoT and edge computing simulation that extends the capability of CloudSim. IoTSim-Edge supports IoT and edge infrastructure and simulates resource heterogeneity, mobility, communication protocol, battery features, and application composition. iFogSim2 [16] is a resource management simulation for IoT, edge, and fog computing. The toolkit supports mobility, migration management, and microservice orchestration. iFogSim2 also supports real mobility datasets and dynamic distributed clustering. Different use cases, such as the crowd-sensing scenario, studio translation scenario, and healthcare scenario. DISSECT-CF-Fog [27] is a simulator built for IoT applications, capable of running large-scale fog and cloud nodes. DISSECT-CF-Fog includes different features such as energy modeling, IoT devices and sensors, mobility, and cost of service analysis. It consists of five modules: simulator, web application, executor, predictor UI, and converter. EdgeSimPy [28] is another Python-based simulator designed for edge computing. EdgeSimPy includes built-in models such as user mobility, application

composition, and power consumption. The core layer of EdgeSimPy includes functional abstractions for data loading, time progression, and entity monitoring. The physical layer includes base stations, network switches, edge servers, and users. The logical layer includes applications, services, container registries, container images, and container layers. The management layer encompasses service placement, service migration, maintenance operations, and network flow scheduling. RayCloudSim [29] is a modeling analytics simulator written in Python for cloud, fog, or edge computing. RayCloudSim can be used for different research purposes, including resource management, scheduling, and task allocation in distributed computing, such as cost analysis in cloud and edge computing. RayCloudSim shares some features with our simulator, like having source code that is simple to read and modify. Additionally, RayCloudSim is easy to integrate with machine learning frameworks. These simulators suit the edge computing environment and do not support the volunteer device's dynamic nature. For volunteer computing simulation, there are a few simulators.

ComBoS [30] is a comprehensive simulation for volunteer computing and desktop grids. ComBoS simulation is a complete scenario of the BOINC project as servers, network, redundant computing, scheduling, and volunteer nodes. ComBos utilizes data from large projects such as ETI@home, Einstein@home, and LHC@home projects for the validation. AVOCLOUDY [31] is a simulator of volunteer clouds. AvoCloudy supports heterogeneity and large scale in highly dynamic systems. The simulation is validated with real data obtained from PlanetLab project. The simulation DENIS [32], a cardiac electrophysiology simulator based on the volunteer DENIS, is a volunteer computing simulation for cardiac electrophysiology tasks. As cardiac electrophysiology is considered a a large task, DENIS tested the task deadline and batch size on the time to complete a batch. Task completion is reduced dramatically from years to days compared to a stand-alone computer.

As we discussed earlier, most existing simulations lack the incorporation of the dynamic environment that volunteer devices bring to the edge environment. Furthermore, the majority of the current simulators are coded in Java or C programming languages, hence restricting their availability to individuals who are skilled in these languages. To the best of our knowledge, VonEdgeSim uniquely simulates the nature of volunteer edge computing for IoT application. Different features, such as energy, dynamicity, with the trust modeling, make it suitable for volunteer edge scenarios. Our solution is Python-based, offering greater flexibility with libraries and ease of development. In Table 1, we compare our simulation with other related work.

Table 1. Comparison of distributed computing simulation frameworks.

Framework	Focus Area	Energy Modeling	Edge Support	Dynamic Resource Availability	Trust Modeling	Implementation
SimGrid, AVOCLOUDY	Volunteer Computing	Limited	No	Yes	Yes	C/Java
CloudSim, DCSim	Cloud Computing Data Centers	Yes	No	No	No	Java
EdgeCloudSim, iFogSim	IoT and Fog Computing	Yes	Yes	No	No	Java
RayCloudSim	Cloud, Fog, Edge Computing	No	Yes	No	No	Python
EdgeSimPy, DISSECT-CF-Fog	Fog-Edge Computing	Yes	Yes	No	No	Python
VonEdgeSim	**Edge and Volunteer Computing**	**Yes**	**Yes**	**Yes**	**Yes**	**Python**

4. Volunteer Edge Computing

Volunteer edge computing is a concept that combines the principles of volunteer computing with edge computing to harness underutilized computing resources at the network's edge for IoT applications.

Volunteer computing: A platform that enables individuals or organizations to contribute their spare resources, such as CPU and memory, to intensive computation tasks, such as scientific projects.

Edge computing: A paradigm that brings computation closer to the edge of the network, near the data users, enhancing processing speed and reducing latency.

In the simulation, where the individual device makes decisions about whether to use its' own resources for the computational task, or to outsource the computations. Our device receives (or generates) some tasks that it should handle. Those tasks have different deadlines and priorities. As our device cannot always handle the tasks it has, it needs to offload some of them. From now on, we will refer to this device as "local device", In order to offload the computations, local device can use one of the three types of helpers:

- A cloud server, which is geographically far from the device, hence requires longer transfer time, but usually has strong computational capability. However, it is worth noting that in general cloud device devotes only a small fraction of its resources to the tasks supplied from our device, as it can potentially be serving lots of other devices simultaneously.
- An edge computer, which is specifically put in a better geographical proximity with the local device for better connection. But it has weaker computational capability compared to the cloud. It will also be serving several other devices.
- A volunteer device, which is similar to the local device, but is idling for some reason, and is ready to provide its' computational capability. On the other hand, volunteer is not always reliable, since at any moment it can go offline, meaning it can stop offering its services and drop all the tasks assigned to it.
- A local device, where the task journey begins. The local device is the user device where a user initiates the task. Then, the task undergoes a decision process on whether to be processed locally or offloaded to the network based on a number of factors, such as energy and delay.

The task distribution mechanism is detailed as follows:

- A task: A task is represented by computational requirements, such as CPU and memory usage, deadline, and priority.
- A device: In the simulators, each device can handle tasks based on its capabilities, computing resources, and proximity to the local device.
- Trust and availability model: Tasks are assigned to volunteers who meet trust logic and score, and are available.
- Optimization: The simulation builds the optimization goal, such as minimizing latency or energy consumption, and performs the simulation of tasks arriving one by one and being assigned to the device, depending on this goal.

There could be various goals that need to be accomplished with respect to the tasks: minimizing the total computational time, minimizing the total execution time, minimizing the total consumed energy, and minimizing the number of failed tasks. We will consider various such setups in the following chapters.

5. VonEdgeSim Design

The simulation is managed by the simulation class. The class constructor takes a network, a list of tasks, and an optimization goal as the parameters, and builds the environment to simulate the task assignment. In this section, we describe all the classes that we developed in the code to set up a proper simulation environment. Figure 1 illustrates the high-level architecture of VonEdgeSim. Each component in the figure corresponds to the subsections discussed below, showing how tasks are generated, managed, and processed within the network system while being monitored by the trust model to ensure system reliability. In Section 5.1, we first discuss the network management with all devices, which is represented as the network manager and general unit in the figure. In Section 5.2, we discuss the simulation class and runtime, which are represented as the simulation engine and runtime control. Then, we discuss the initialization and generation of resources, represented by the task generator in the figure. Finally, in Section 5.3, we present the trust model, which is represented by the trust model in the figure.

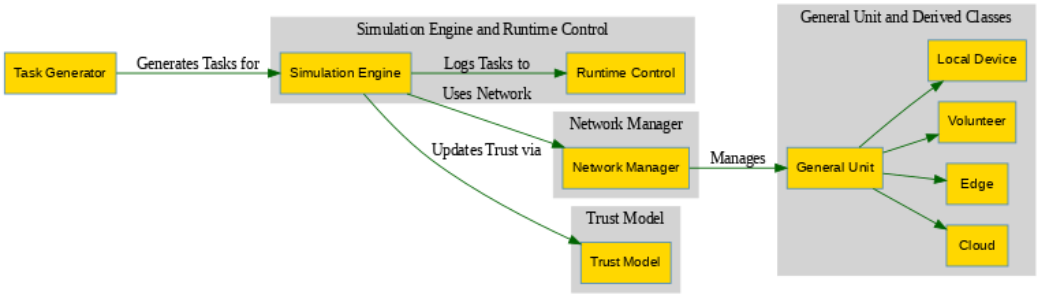


Figure 1. High-level architecture of VonEdgeSim.

5.1. The Network

The network is a set of devices (cloud, edges, volunteers and local device). To construct an object of this class, we need to pass to the contructor a list for each device type that constitutes the network. In Table 2, we list the important notation used in simulator. The general unit serves as the foundation for all the devices within the network. It contains the characteristics that are shared by all devices.

Table 2. Notation table.

Symbol	Description
name	A string containing the name of the device.
ℓ	Distance over which the data will have to be transmitted tirelessly to reach this device [m].
f	CPU speed [MHz].
c	Price for using the device.
B	Connection speed the device has with respect to the local device [Mbit/s].
τ	Latency of the network between the given device and the local device [s].
T_c	Computation time of a task [s].
C_{task}	Number of cycles required for executing the task $[Million\,cycles]$.
T_d	Time for the data transfer [s].
M_{task}	Task's memory size [Mbit].
T	Trust score.
w_1, w_2, w_3	Weights assigned to each term in the trust formula.
S	Success rate.
L	Average latency [s].
C	Average completion time [s].
E_{comp}	Computation energy [Wh].
T_{comp}	Computation time [s].
P_{CPU}	CPU energy usage [W].
E_{transfer}	Data transfer energy [Wh].
P_{bw}	Bandwidth energy usage [W].
T_{transfer}	Data transfer time [s].
R_{wireless}	Wireless transmission rate [Mbit/s].
P_{trans}	Transmission power [mW].
h	Channel fading coefficient (wireless transmission).
path loss	Path loss coefficient based on distance and path loss exponent.

5.1.1. Execution Time and Energy Model

The total execution time T_{total} is calculated by summing the computation, data transfer, and wireless transfer times. This is the primary factor used to determine the best device for task execution.

$$T_{\text{total}} = T_{\text{comp}} + T_{\text{transfer}} + T_{\text{wireless}}$$

where

$$T_{\text{comp}} = \frac{C_{\text{task}}}{f}$$

is the computation time, with C_{task} representing the number of CPU cycles required for the task and f being the CPU speed.

$$T_{\text{transfer}} = 2(\frac{M_{\text{task}}}{B} + \tau)$$

is the data transfer time, where M_{task} is the memory size of the task, B is the bandwidth, and τ is the latency, The factor of 2 accounts for the round-trip time, meaning the time for sending the data and receiving a response.

$$T_{\text{wireless}} = \frac{M_{\text{task}}}{R_{\text{wireless}}} + \tau$$

is the wireless transfer time. The wireless transmission rate R_{wireless} is computed using the following formula:

$$R_{\text{wireless}} = B \cdot \log_2\left(1 + \frac{P_{\text{trans}} \cdot h}{\text{noise} \cdot \text{path loss}}\right)$$

where P_{trans} is the transmission power, h is the channel fading coefficient, and path loss is a function of the distance and path loss exponent.

Energy Consumption

The total energy consumption E_{total} for a task is calculated as the sum of computation energy and data transfer energy. Energy is tracked as part of the task execution process, allowing insights into the energy requirements for different devices.

$$E_{\text{total}} = E_{\text{comp}} + E_{\text{transfer}}$$

where

$$E_{\text{comp}} = P_{\text{CPU}} \cdot T_{\text{comp}}$$

represents the computation energy, and

$$E_{\text{transfer}} = P_{\text{bw}} \cdot T_{\text{transfer}}$$

represents the energy consumed during the data transfer.

5.1.2. Device Classes

The device classes in VonEdgeSim are all children of the general unit class, each with specific characteristics tailored to their roles within the network.

Local device: The local device class is a child of the general unit class. It has the location and cost rate hardcoded to 0, with the name set to "myDevice". In the simulation, a single local device is initialized, representing the source of task generation. The local device's processing capabilities, including CPU speed and energy consumption, are defined within the simulation environment.

Cloud: The cloud class is another child of the general unit, with hard-coded parameters for wireless transmission.

Edge: Similar to the cloud class, the edge class is a child of the general unit, also featuring hard-coded parameters for wireless transmission.

Volunteer: The volunteer class, also derived from the general unit class, has hard-coded parameters for wireless transmission. Additionally, it includes the following parameters: trust, which is a value between 0 and 1 representing the trustworthiness of the volunteer based on its previous behavior; availability, which is a Boolean value with 0 indicating unavailability; last shutdown time, which is the time when the volunteer last went offline; and the number of executed tasks, representing the number of tasks that the volunteer completed successfully.

5.1.3. Tasks Classes

The set of tasks is represented as a list of task class objects. The task class contains the description of the task, which arises at the local device and needs to be assigned to one of the units. The variables in the task class include the id, which is a unique identifier for the task; the type, which is a string used to classify the task into categories; the arrival time, which is the time when the task appears at the local device. If the task is reassigned, the arrival time changes to the reassignment time, but the wasted time is tracked. The deadline time is the time after which the task is considered failed unless it is completed successfully before it. The CPU cycles refer to the number of cycles required to execute the task, and the memory size is the size of the input data that must be transmitted to the computing device for task execution. The start time is when the task's execution begins, and the end time is when the execution ends. The wasted time is the duration between the task's first appearance and when its execution starts. The assigned unit is the unit that executes the task. The success flag is a Boolean variable indicating if the task was completed successfully (1) or not (0). The assignment counter keeps track of the number of times the task was assigned to a unit (greater than 1 if the task was reassigned due to volunteer shutdown). Finally, the deadline specifies how much time a task has to be finished.

5.2. The Simulation Class

The simulation class takes the network and tasks as arguments, along with the optimization goal, and performs the simulation of tasks arriving one by one and being assigned to the appropriate device, depending on the optimization goal. The variables of the simulation class include the network, which is an object of the network class containing all the units; the tasks, which are a list of tasks sorted by arrival time, with the sorting performed automatically by the constructor of this class (meaning the user does not need to worry about sorting the tasks); the not assigned tasks, which is an initially empty list that will be filled with the tasks that could not be assigned to any unit (most likely due to an unreachable deadline); and the runtime control, which is an item of a log runtime control class.

5.2.1. Simulation Framework Variables

In this section, we discuss different variables used within the simulation framework. The volunteer trust threshold sets the minimum trust level that a volunteer device must meet to be ready for task. The simulation length represents the total duration of the simulation in seconds, establishing the temporal context for task generation and processing, and influencing the simulation's scope and the volume of data handled. The volunteer timeout threshold determines the duration after which volunteers are considered unavailable to accept any coming tasks. The transition coefficient is a parameter used for transforming energy cost into time cost.

Runtime control: We created this task to keep track of task assignments and provide a general overview of the simulation. The runtime control includes lists such as assigned tasks, which are the tasks that have been assigned; units, which are the units to which each task is assigned; task index, a list of tuples with the task ID as the first element and the position of the task in the assigned tasks list as the second element, used to find a task and the related unit by knowing its ID; and not assigned tasks, which are the tasks that could not be assigned because no device could meet their deadline.

5.2.2. Random Initialization

Although for some debugging purposes, we create some fixed devices or tasks, for the final simulation, we generate tasks and networks randomly. The task generator is a function that takes the desired number of tasks as an argument and returns a list of tasks. We also use the global variable simulation length to control the total duration of the simulation. Second, the network generator is used to generate a network with a specified number of clouds and edges. As we want to compare networks without volunteers to those with

volunteers, we first generate a random network without volunteers, and then separately implement the function that adds random volunteers to the network. Finally, we generate volunteers. Unlike previous functions, which were independent and created class objects; this function is part of the class and expands an existing class object. It is a short function that calls an individual generator, but we present it here in a unified form.

5.3. Trust Model

Trust Calculation Formula

The trust score (T) is calculated as follows:

$$T = w_1 \cdot S + w_2 \cdot \left(\frac{1}{1+L}\right) + w_3 \cdot \left(\frac{1}{1+C}\right) \tag{1}$$

where S represents the success rate, L represents the average latency, and C represents the average completion time. Here, w_1, w_2, and w_3 are the respective weights assigned to each term.

The first term in the formula, success rate (S), is defined as follows:

$$S = \frac{\text{successful Tasks}}{\text{successful Tasks} + \text{failed Tasks} + 1} \tag{2}$$

This formula calculates the ratio of completed tasks to the total number of tasks, with an added one to avoid division by zero, inspired by well-known trust models [9,33].

The second term, average latency (L), is calculated as follows:

$$L = \frac{\text{total Latency}}{\text{successful Tasks} + 1} \tag{3}$$

This represents the average latency for each successfully completed task. The total latency is calculated as the sum of the latencies of all successfully completed tasks.

The third term, average completion time (C), is given by

$$C = \frac{\text{total Completion Time}}{\text{successful Tasks} + 1} \tag{4}$$

This calculates the average time consumed by the successful tasks. In the simulation, the trust terms are managed during task assignment with volunteer trust threshold to filter trusted volunteers that meet the threshold.

The volunteer timeout threshold determines the duration after which a volunteer is considered inactive. If a volunteer's trust score drops below the volunteer trust threshold and they remain inactive for a period exceeding the volunteer timeout threshold, the system restores the volunteer's trust, making them ready for task assignment again.

5.4. Diagram

In the Figure 2, we present the UML sequence diagram, which provides a high-level overview of the simulation's flow and components. The VonEdgeSim simulation's workflow is illustrated in this figure. The simulation begins with the user creating tasks and instantiating the network, which includes different types of devices such as volunteers, cloud, edge, and local devices. The figure shows task generation and assignment. Additionally, a trust score check is performed on volunteer devices before tasks are assigned. Tasks are distributed to the appropriate device (cloud, edge, volunteer, or local device) as the simulation iterates through them. The diagram highlights key components such as the network, task management, runtime control, and their interactions.

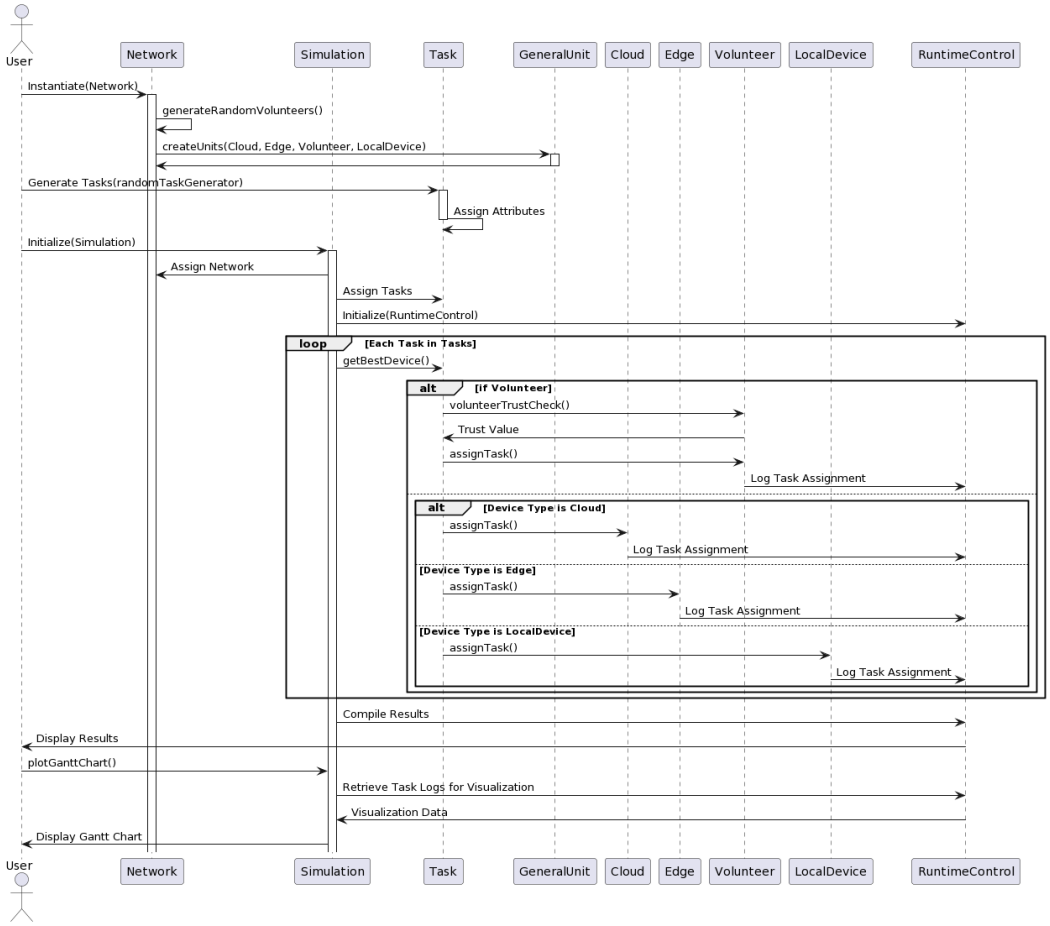


Figure 2. The diagram simulation's flow and components.

6. The Implementation

After the network, tasks, and the corresponding simulation object are created, we can run the simulation. It simulates assigning the tasks according to the criteria that the user selected. The simulation class not only assigns the tasks to the different devices but also decides the temporary shutdowns and turn-ons of volunteers. In order to control the frequency at which those occur, we use the variables shutdown frequency and turn-on frequency. We will explain their usage later in this section. To avoid the volunteers' shutdown or turn-on, we can set the corresponding shutdown and turn-on values to 1. The simulation is run by calling the run with the random shutdowns function, which oversees the process of randomly shutting down and turning on volunteers, triggering the re-assignment process as necessary. The get best device, shutdown volunteer, and run with random shutdowns together ensure that when a volunteer becomes unavailable, tasks are seamlessly redistributed to edge devices or other volunteers, maintaining the integrity and efficiency of the distributed system.

Now, we describe step by step what the code does to run the simulation:

- Start the runtime control object to start collecting printable data (Line 1).
- For a task in the list of tasks (already sorted by arrival time) (Line 2):
  - if the condition on Line 3
    (i.e., if the remainder of dividing task IDs by the shutdown frequency) is 1.
    We use this to have a frequency for the shutdowns of volunteers. We decide with this on average after how many tasks should a random volunteer shutdown. Since tasks IDs are not sequenced (as tasks were sorted by arrival time, their indexes are not arranged) the shutdowns occur at random times, but on average after every shutdown task arrives in my device.

* Obtain a list of active volunteers and if this list is not empty, randomly choose one of them and call the shutdown volunteer function to simulate the shutdown of the randomly selected volunteer at the current task's arrival time (Line 5–7).
  In this function, following steps are taken:
  - Collect all tasks that will need to be reassigned, by calling the volunteer class member function volunteer.shutdown. For the selected volunteer, conduct the following:
  - Set the availability to 0.
  - Set the last shutdown time to the current shutdown time.
  - For the tasks in the device's queue, collect all whose end time is after the shutdown time (in order to send them for reassignment), and then delete them from the volunteer's log.
  - Adjust the trust by

$$T = w_1 \cdot S + w_2 \cdot \left(\frac{1}{1+L}\right) + w_3 \cdot \left(\frac{1}{1+C}\right) \tag{5}$$

    two is added here to avoid setting the trust to zero when no tasks were completed before shutdown, and also to balance the penalty when few tasks were assigned before the shutdown.
  - Now, back in the original shutdown volunteer function, to handle the reassignment of tasks from a volunteer that has been shutdown to another available device, cycle through all the tasks that should be reassigned and
  - prepare the task to reassign by calling task.prepare. For reassignment function (line 8–9)
  - set wasted time to previously wasted time plus time spent in the queue of the last volunteer. More precisely

$$wastedTime = wastedTime + reassignmentTime - arrivalTime.$$

  - Reset the arrival time to the reassignment time.
  - Clear the assigned unit, leaving it unassigned.
  - Initialize the start time to an undefined state.
  - Initialize the end time to an undefined state.
  - Find the new device for this task by calling get best device function. This function handles the distribution of tasks to the most suitable device, be it a volunteer, edge, or cloud, based on availability and other factors. As this function is dependent on what we are optimizing for, for now the only important thing to keep in mind is that it returns the best device key and a success flag for finding the device that could fulfill the constraints.
  - If the device is successfully found, call the best device function (Line 10).
  - Compute execution time of the task on this device by calling compute execution time function (see this function in the unit class description).
  - If the device is not doing any tasks
  - set the task's end time to the arrival time plus the execution time.
  - Otherwise, If the device is currently executing another task, set the device's end time to device's end time + execution time.
    In other words, if the device already has other tasks, it will start executing this task only after it is finished with all the previous ones in the queue.
  - Otherwise, if no device could satisfy the tasks deadline, we add the task to the list of failed tasks (Line 14).

- After the shutdown happened, or the condition was false and shutdown was not needed, the function continues to process the current task by finding the best device for it, using the function get best device (Line 19).
  Note that if the shutdown happened during the previous step, then this volunteer will not be among the devices to choose from, as its availability would be switched to 0 and it will not pass the availability check.
- If finding the best device ended successfully, we call the best device function again. otherwise we append the task to the list of failed tasks (Line 20–24).
  Note that this step is basically the same as what was happening during the reassignment.
- For the condition in line 25.
  (i.e., if the reminder of dividing task IDs by the turn-on frequency is 1, using the same mechanics as for shutdown) (Line 25–28).

The simulation steps and VonEdgeSim pseudo-code is shown in Algorithm 1.

Algorithm 1 Simulation Steps

```
1:  Initialize runtimeControl object for data collection
2:  for each task_i in tasks sorted by arrivalTime do
3:    if task_i.ID mod shutDownFreq == 1 then
4:      activeVolunteers ← getListOfActiveVolunteers()
5:      if activeVolunteers is not empty then
6:        selectedVolunteer ← selectRandom(activeVolunteers)
7:        shutDownVolunteer(selectedVolunteer, currentTaskArrivalTime)
8:        for each task in selectedVolunteer.tasks needing reassignment do
9:          task.prepareForReassignment(currentTaskArrivalTime)
10:         (bestDevice, success) ← getBestDevice(task)
11:         if success then
12:           bestDevice.assignTask(task)
13:         else
14:           append task to failedTasks list
15:         end if
16:       end for
17:     end if
18:   end if
19:   (bestDevice, success) ← getBestDevice(task_i)
20:   if success then
21:     bestDevice.assignTask(task_i)
22:   else
23:     append task_i to failedTasks list
24:   end if
25:   if task_i.ID mod turnOnFreq == 1 then
26:     performTurnOnOperations()
27:   end if
28: end for
```

7. Experiments

In this section, we discuss the experiment in detail. First, we introduce the benchmark case and simulator environment. Next, we present the simulation results. Finally, we summarize the key lessons learned from the simulation experiment. The purpose of the experiments is to demonstrate how the previously mentioned research issue is effectively addressed by the proposed simulation framework, particularly in the context of improving task allocation and execution in volunteer edge computing environments. Our goal is to provide clear evidence of the framework's effectiveness by linking the experimental results to our scientific contributions.

7.1. Simulators Parameters

We describe the design of a benchmark case. Tables 3 and 4 define the network and task parameters [14]. In Table 3, we present the network parameter values for all network units, including the cloud, edge, volunteer, and local device. Parameters such as number, CPU speed, bandwidth, latency, energy consumption, and bandwidth energy consumption are provided for each network unit.

Table 3. Network parameters overview.

Network Parameters Overview				
	Cloud	**Edges**	**Volunteers**	**Local Device**
Number	1	3	0–100	1
CPU Speed (MHz)	3600–3900	2200–3500	1000–2500	1000–2000
Bandwidth (Mbit/s)	50–100	50–100	50–867	10–50
Latency (s)	0.01–0.08	0.001–0.01	0.001–0.1	0.0
Cycle Energy Consumption (Wh)	100–150	75–150	75–125	75–125
Bandwidth Energy Consumption (Wh/Mbyte)	175–225	150–200	100–150	75–125

Table 4 illustrates three different task types: Augmented Reality (AR), Health Monitoring (HM), and Infotainment (I). We present parameters such as interarrival time, CPU cycles, memory size, and deadline. Interarrival time affects the rate at which tasks are generated, where shorter interarrival times increase system load. CPU cycles (measured in mega instructions) represent the computational requirements of each task, with higher cycles demanding more processing time and energy.

Table 4. Task parameters.

Tasks Parameters			
Parameter Name	**Augmented Reality (AR)**	**Health Monitoring (HM)**	**Infotainment (I)**
Interarrival Time (s)	$\sim \mathcal{E}(2)$	$\sim \mathcal{E}(5)$	$\sim \mathcal{E}(10)$
CPU Cycles (Mega Instructions)	20,000	7500	2500
Memory Size (kB)	1500	25	1250
Deadline (s)	300–900	180–600	60–300

In Table 5, we present additional parameters and simulation settings such as the number of simulation repetitions, duration, number of tasks, along with transmission power, fading, path loss, and noise power.

7.2. Simulators Results

In the current version of the code, we perform five tests to demonstrate the usage and capabilities of the simulation. These tests include the following: (a) testing the task distribution among the network units; (b) testing delay, task failure, and execution time as the number of tasks increases; (c) testing delay, task failure, and execution time as the number of volunteers increases; (d) analyzing the performance of different task types, and finally; (e) testing the time and memory complexities of the simulation.

7.3. Task Distribution

Initially, we carry out comparative simulations of network distribution, both with and without volunteers, as depicted in Figures 3 and 4. A randomly generated network with a set of randomly generated tasks is tested. The network consists of 1 cloud, 3 edge servers,

5 volunteers, and 2000 tasks. The second test uses the same parameters but excludes volunteers. In Figure 3, the task distribution across the network units is as follows: cloud hosts 476 tasks, edge hosts 804 tasks, volunteers host 664 tasks, and the local device runs 56 tasks. A different distribution is shown in Figure 4, where the cloud hosts 39 tasks, edge hosts 1724 tasks, and the local device handles 237 tasks. In this small test, we do not evaluate performance metrics such as delay or task failure; rather, we illustrate how our simulation distributes the tasks among the network units. Figure 5 compares the energy consumption for the two scenarios: (1) with volunteers and (2) without volunteers. It is clear that when volunteers are involved, the energy consumption of the cloud and local device drops significantly. The energy consumption for each network unit increases based on the energy required for each task, which includes data transfer and computation. During the evaluation, total energy consumption increases based on the nature of the tasks. Additionally, advanced task assignment and load balancing are beyond the scope of the current version of the simulation.

Table 5. Simulation parameters overview.

Simulation Parameters Overview	
Simulation Repeats	5 times
Simulation Duration	21,600 s (6 h)
Number of Tasks	10,000
Shutdown Frequency	10
Turn-On Frequency	100
Volunteer Trust Threshold	0.3
Volunteers Timeout Threshold	Simulation length × 0.1 s
Transmission Power	10 dBm (converted to 10 mW)
Fading Coefficient	Random value between 0.5 and 0.9
Path Loss Exponent	Random value between 2 and 3.5
Noise Power	0.001

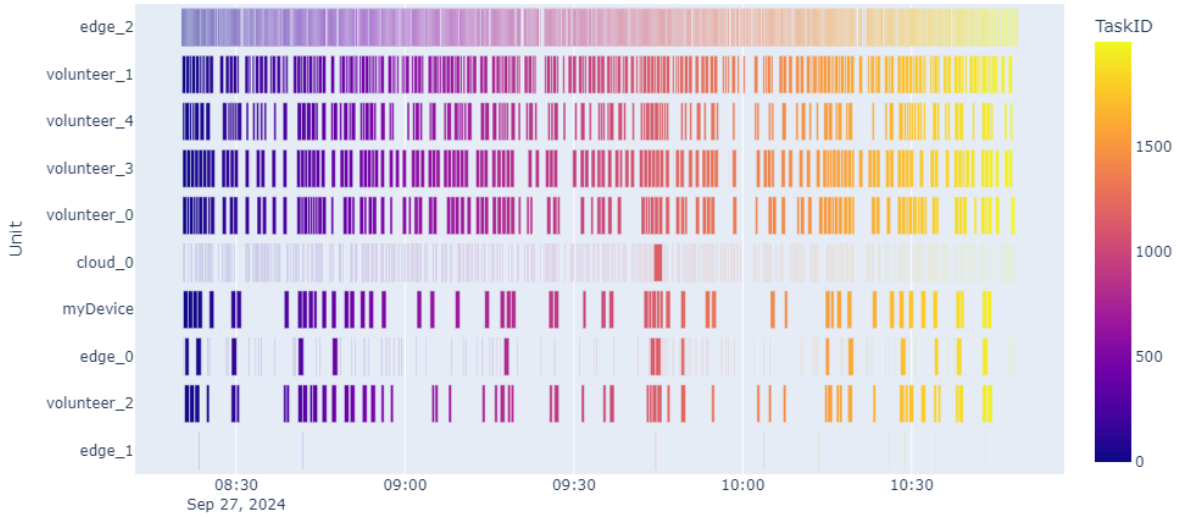


Figure 3. Simulation of a network with 1 cloud, 3 edges, and 5 volunteers. A total of 2000 tasks were generated.

7.4. Testing Performance Metrics with Increasing Task Numbers

In Figure 6, the second test compares delay, task failure, and execution time as the number of tasks increases across different scenarios. The network distribution consists of

1 cloud, 3 edge servers, and 100 volunteers, with tasks increasing up to 100,000, as shown in Tables 3 and 5. The first scenario is called TrustTrue, where the model uses the trust model in the assignment with volunteer resources. The second scenario is called TrustFalse, where the model does not use the trust model in the assignment with volunteer resources. In some parts of the discussion, this scenario may also be referred to as untrust for simplicity. The final scenario, called NoVolunteer, involves no volunteer resources in the assignment. In the final scenario, only one cloud and three edge servers are available to host the tasks.

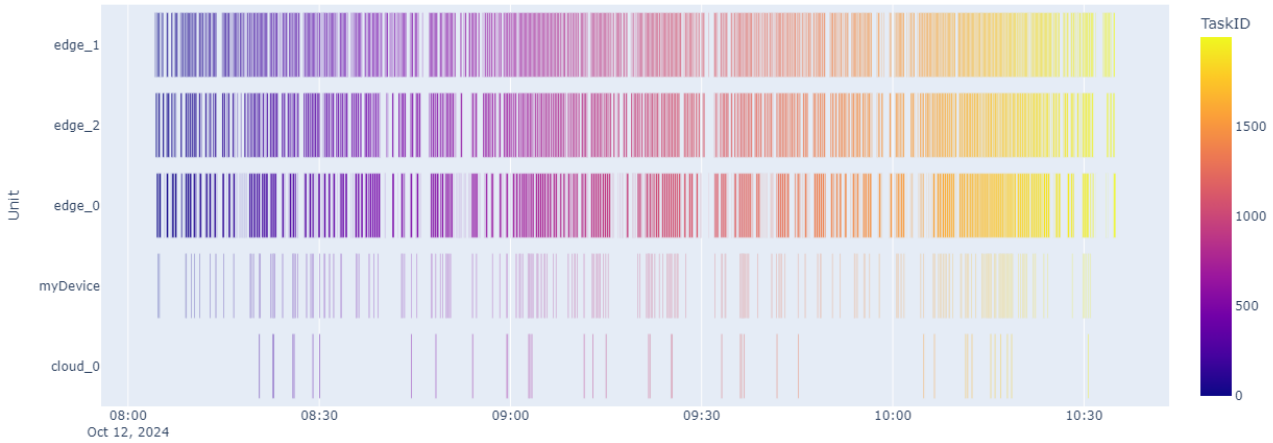


Figure 4. Simulation of a network with 1 cloud, 3 edges, and 0 volunteers. A total of 2000 tasks were generated.

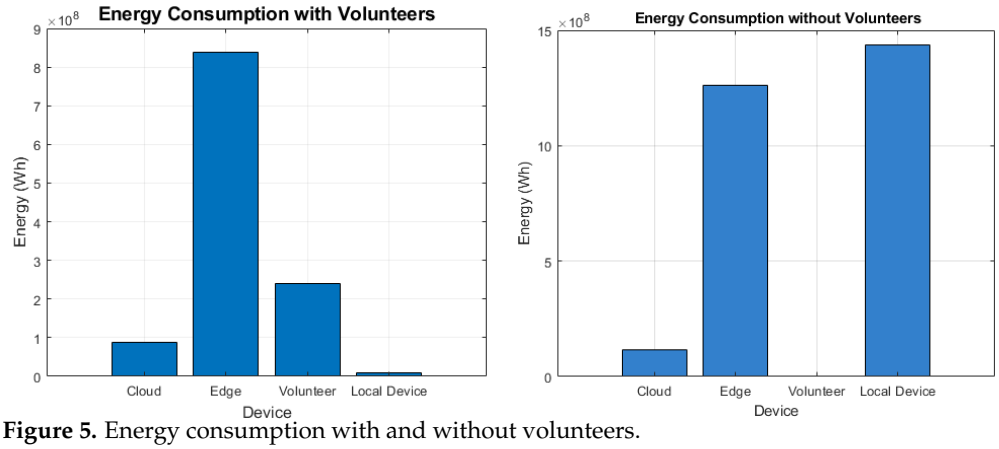


Figure 5. Energy consumption with and without volunteers.

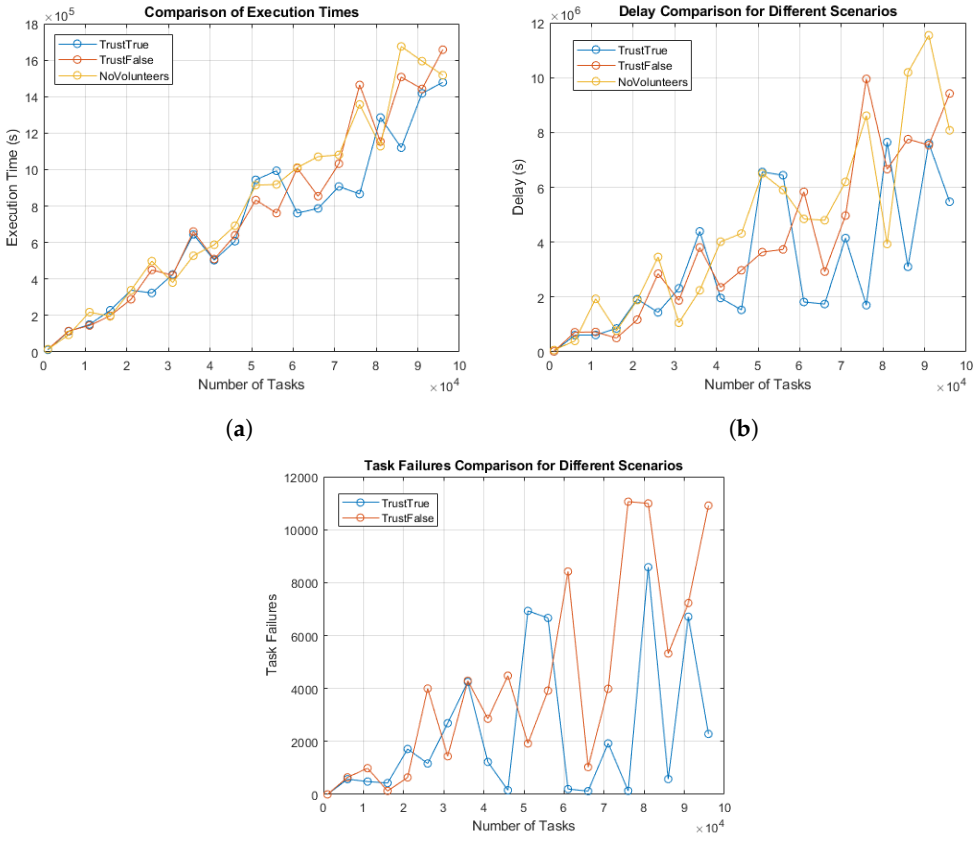
In Figure 6a, we compare the execution time between scenarios. It is clear that as tasks increase, the system load increases, causing the execution time to rise. It is noticeable that the TrustTrue scenario outperforms the other scenarios in most observations. The trust model outperforms the TrustFalse model by 5%, and by 9% in comparison to the NoVolunteer scenario. This shows the promising potential of using volunteers to support edge and cloud environments, with a 9% improvement, which is significant. The reason we see faster execution times in the TrustTrue scenario is that the trust model relies on the history of volunteers' completion times and prefers those volunteers who have lower average completion times.

However, there are a few observations of spikes in the TrustTrue scenario, where the other models show better results. These spikes are related to both the nature of the task types (specifically the high demands and frequency of AR tasks) and how the trust model handles those demands. Since AR tasks are frequent and resource-intensive, they can cause resource contention and fluctuations in trust. Addressing these observations requires an advanced and scalable task assignment model, where the trust model is further utilized and optimized.

In Figure 6b, we compare the delay between scenarios. It is evident that as tasks increase, the system experiences more load, leading to a rise in delay time. It is clear that the trust model exceeds other scenarios in most cases. The trust model surpasses the TrustFalse model by 6%, and by 16% compared to scenarios without volunteer involvement. This shows the promise of utilizing volunteers in an edge environment. The system struggles more significantly under larger loads without volunteer support. It clearly demonstrates that the trust model decreases delay compared to the experiment without trust because the nature of modeling trust prioritizes volunteers with better performance, making them more likely to be chosen.

Volunteers with a trust score below the volunteer trust threshold (*Volunteer Trust Threshold = 0.3*) are considered unreliable and are therefore excluded from task assignment, which effectively reduces the chances of task failures and consequently decreases latency. However, further investigation into the anomalies and the behavior of the system in certain observations would be beneficial to understand and mitigate the observed spikes in delay.

In Figure 6c, where we compare only between the trust and untrust model in task failure. The results show that trust model outperform the untrust model with 32%. During the task assignment, we set the volunteer trust threshold and volunteer timeout threshold to control whether the volunteer is ready for assignment or if his trust has fallen below the threshold. The result shows that the result is better with trust enabled, relying on volunteer history and the dynamicity management of this value during the assignment to improve the reliability system. The trust model calculates both the number of successful and failed tasks over time, thus favoring volunteers with a high number of successful tasks.



(**a**) (**b**)

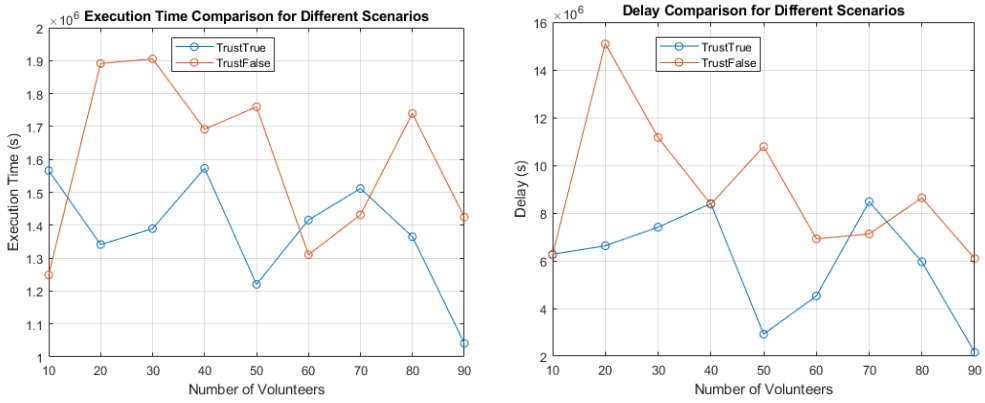
(**c**)

Figure 6. Execution time, task delay, and task failure, as the number of tasks increases. (**a**) Execution Time. (**b**) Task Delay. (**c**) Task Failure.

7.5. Testing Performance Metrics with Increasing Volunteer Numbers

In Figure 7, the second test compares the trust and untrust models in terms of delay, and execution time as the number of volunteers increases. Additionally, the test shows how volunteer resource involvement can support edge computing. The network distribution consists of one cloud, three edge servers, task tasks is 100,000, and volunteers increase from 10 to 90.

In Figure 7a, we compare the execution time between scenarios. It is obvious that as the number of volunteers increases, the availability of more computational resources leads to a decrease in execution time. It is also noticeable that the trust model outperforms the untrust model in most observations. The trust model outperforms the untrust model by 13%, and the execution time improves by 33% as the number of volunteers increases from 10 to 90. In Figure 7b, we compare the delay time between trust and untrust models. It is clear that as the number of volunteers increases, the delay decreases because more computational resources are involved in handling the tasks. The results show that the trust model outperforms the untrust model by 34%, and the delay time improves by 64% as the number of volunteers increases from 10 to 90. The main objective of volunteer edge computing is to support edge computing, the results show promising outcomes in decreasing delay and execution time. As we discussed previously, the spikes in some observations, such as when the number of volunteers is 40 and 70, are because we ran the simulation with different types of tasks within different intervals. For example, the task type 'AR' is more frequent and consumes more delays. Further discussion on evaluating task types is covered in the next section.



(**a**) (**b**)

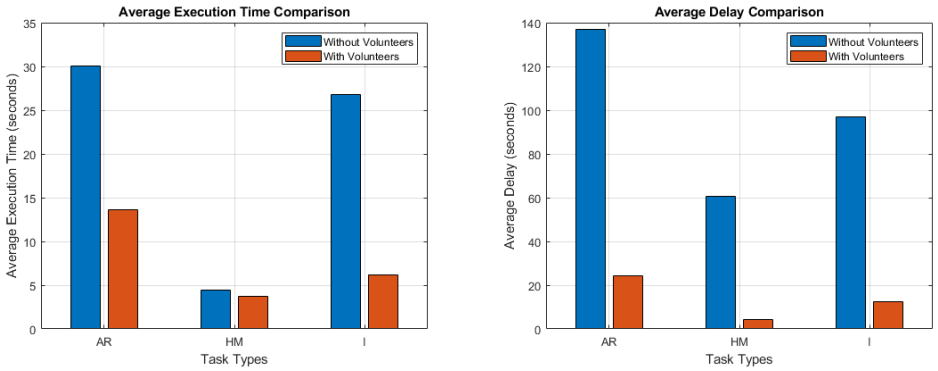
Figure 7. Execution time and task delay as the number of volunteer increases. (**a**) Execution time. (**b**) Task delay.

7.6. Task Type Performance Analysis

In this section, we evaluate the performance of the simulation on task types with and without volunteers. We compare the results of execution time and delay in both scenarios. The network setup consists of a single cloud, 3 edge servers, and 100,000 tasks, with 50 volunteer resources available. In Figure 8, we illustrate the delay and execution time with and without volunteers. Both execution time and delay decrease when volunteer resources are involved.

The nature of each task differs, as shown in Table 4. For instance, AR (Augmented Reality) tasks are more frequent and resource-intensive than other task types. Without volunteers, the cloud and edge servers struggle to meet the demands of these high-resource tasks. In contrast, HM (Health Monitoring) tasks, which are less computationally demanding compared to AR, can be handled by the cloud and edge, but the delay increases without volunteer support. Lastly, I (Infotainment) tasks show significant improvement with the addition of volunteers, reducing both delay and execution times. These results

highlight the substantial benefits of adding volunteers to enhance the performance of edge computing environments.



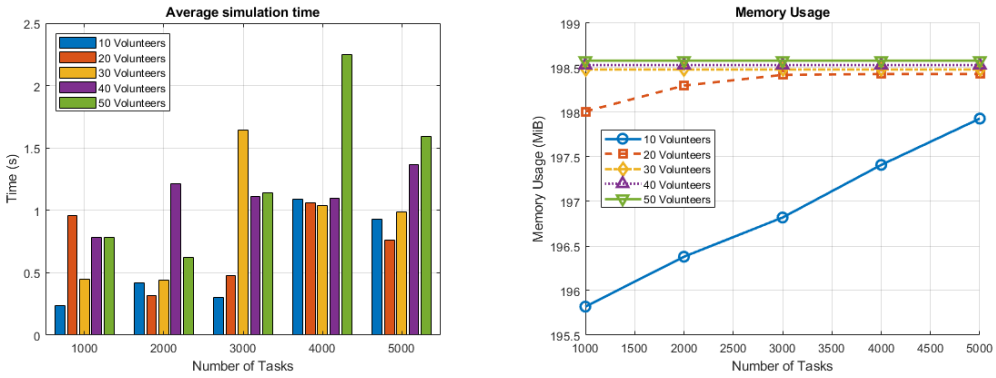
(**a**) (**b**)

Figure 8. Execution time, and task delay (with and without volunteers). (**a**) Execution time. (**b**) Task delay.

7.7. Time and Memory Complexities

Additionally, we analyze the scalability of the simulation in terms of simulation time and memory usage. We conducted experiments with tasks ranging from 1000 to 5000, across different numbers of volunteer edges, from 10 to 50. Figure 9a shows that the tests on simulation time demonstrate that as the load increases, the simulation time is not dramatically affected. Running 5000 tasks over 50 volunteers is completed in 1.59 s, which is a very reasonable time, with a difference of only 0.66 s compared to running the test with 10 volunteers.

Regarding memory usage, the results in Figure 9b show consistency across all cases. The simulation of 5000 tasks with 50 volunteers consumes 198.38 MiB, with only a 0.45 MiB difference compared to running the test with 10 volunteers.



(**a**) (**b**)

Figure 9. Scalability of the simulation. (**a**) Simulation time. (**b**) Memory usage.

To sum up, Table 6 is dedicated to presenting the main lessons learned in the work. The experiment discusses the effectiveness of using volunteer devices in edge computing environments to improve task execution, and delay reduction. In conclusion, this series of experiments is conducted to validate the simulation. The result shows that integrating the volunteer edge resources in edge computing can improve the performance of edge solutions.

Table 6. Summary of main lessons learned.

Lesson Learned	Key Observations
Reduction in Task Delay	Volunteers help decrease total task delay by nearly 64%, demonstrating the effectiveness of distributed resources.
Reduction in Task Execution	Volunteers involvement help decrease total task execution by nearly 33%, demonstrating the benefit of volunteer resources.
Complexities of the Simulation	The simulation handles increased numbers of volunteers and tasks efficiently, with minimal impact on execution time and memory usage.
Effectiveness of the Trust Model	The trust model reduces latency, improves task completion times, and lowers task failure rates.

8. Conclusions and Future Works

The volunteer edge computing infrastructure offers highly scalable capabilities. However, due to the challenges posed by volunteer resources, users cannot easily deploy their APIs and tools on this infrastructure. Primary concerns, such as availability and trust, remain significant issues within volunteer infrastructure. Moreover, testing the proposed framework or hypothesis in real systems incurs substantial costs. Therefore, this paper presents a simulation for volunteer edge computing. To the best of our knowledge, this is the first paper to include volunteer edge devices in a simulation. We have conducted several tests to demonstrate VEC's ability to advance edge computing. We ran well-known IoT applications, including Augmented Reality, Infotainment, and Health Monitoring. The results demonstrate the simulation's capability to significantly reduce total task delay and task failures, and improve task execution time. This simulation establishes a foundation for future research to enhance VEC and address the challenges, setting the stage for the development of more reliable, effective, and scalable solutions in edge computing. Our future goals include expanding the network models and protocols for a comprehensive analysis of delays and integrating vehicles along with their characteristics into this simulation. Additionally, our future work will cover URLLC aspects, supporting ultra-reliable and low-latency communications for 5G applications. Moreover, incorporating concepts such as context-aware computing and context histories could further enhance the adaptability and intelligence of volunteer edge computing systems [34,35]. By leveraging context prediction and similarity analysis, future simulations could better anticipate system states and optimize task management. Finally, the reward functionality is not implemented in this work and will be included in future development. Code is available at https://github.com/SimuEnv/VolunteerEdgeSim (accessed on 10 October 2024).

Funding: This work was supported by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under Grant GPIP: 595-611-2024.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The author declare no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

References

1. Qiu, T.; Chi, J.; Zhou, X.; Ning, Z.; Atiquzzaman, M.; Wu, D.O. Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2462–2488. [CrossRef]
2. Yousefpour, A.; Fung, C.; Nguyen, T.; Kadiyala, K.; Jalali, F.; Niakanlahiji, A.; Kong, J.; Jue, J.P. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.* **2019**, *98*, 289–330. [CrossRef]
3. Mengistu, T.M.; Che, D. Survey and taxonomy of volunteer computing. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 1–35. [CrossRef]

4. Ma, P.; Garg, S.; Barika, M. Research allocation in mobile volunteer computing system: Taxonomy, challenges and future work. *Future Gener. Comput. Syst.* **2024**, *154*, 251–265. [CrossRef]
5. Alshuaibi, E.A.; Hamdi, A.M.; Hussain, F.K. Volunteer Computing for fog scalability: A systematic literature review. *Internet Things* **2024**, *25*, 101072. [CrossRef]
6. Alsenani, Y.; Alnori, A. Trust-aware Scheduling for Edge Computing with Task Dependencies and Unreliable Servers. *IEEE Access* **2023**, *11*, 113514–113525. [CrossRef]
7. Yeddulapalli, H.S.; Alarcon, M.L.; Roy, U.; Neupane, R.L.; Gafurov, D.; Mounesan, M.; Debroy, S.; Calyam, P. VECA: Reliable and Confidential Resource Clustering for Volunteer Edge-Cloud Computing. *arXiv* **2024**, arXiv:2409.03057.
8. Alsenani, Y.; Crosby, G.V.; Velasco, T.; Alahmadi, A. ReMot reputation and resource-based model to estimate the reliability of the host machines in volunteer cloud environment. In Proceedings of the 2018 IEEE 6th international conference on future internet of things and cloud (FiCloud), Barcelona, Spain, 6–8 August 2018; pp. 63–70.
9. Alsenani, Y.S.; Crosby, G.V.; Ahmed, K.R.; Velasco, T. ProTrust: A probabilistic trust framework for volunteer cloud computing. *IEEE Access* **2020**, *8*, 135059–135074. [CrossRef]
10. Alsenani, Y.; Crosby, G.; Velasco, T. SaRa: A stochastic model to estimate reliability of edge resources in volunteer cloud. In Proceedings of the 2018 IEEE international conference on EDGE computing (EDGE), San Francisco, CA, USA, 2–7 July 2018; pp. 121–124.
11. Mansouri, N.; Ghafari, R.; Zade, B.M.H. Cloud computing simulators: A comprehensive review. *Simul. Model. Pract. Theory* **2020**, *104*, 102144. [CrossRef]
12. Jha, D.N.; Alwasel, K.; Alshoshan, A.; Huang, X.; Naha, R.K.; Battula, S.K.; Garg, S.; Puthal, D.; James, P.; Zomaya, A.; et al. IoTSim-Edge: A simulation framework for modeling the behavior of Internet of Things and edge computing environments. *Softw. Pract. Exp.* **2020**, *50*, 844–867. [CrossRef]
13. Boakye-Boateng, K.; Ghorbani, A.A.; Lashkari, A.H. A Trust-Influenced Smart Grid: A Survey and a Proposal. *J. Sens. Actuator Netw.* **2022**, *11*, 34. [CrossRef]
14. Sonmez, C.; Ozgovde, A.; Ersoy, C. Edgecloudsim: An environment for performance evaluation of edge computing systems. *Trans. Emerg. Telecommun. Technol.* **2018**, *29*, e3493. [CrossRef]
15. Buyya, R.; Ranjan, R.; Calheiros, R.N. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In Proceedings of the 2009 International Conference on High Performance Computing & Simulation, Leipzig, Germany, 21–24 June 2009; pp. 1–11.
16. 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. [CrossRef]
17. Gonzalo, S.; Marquès, J.M.; García-Villoria, A.; Panadero, J.; Calvet, L. CLARA: A novel clustering-based resource-allocation mechanism for exploiting low-availability complementarities of voluntarily contributed nodes. *Future Gener. Comput. Syst.* **2022**, *128*, 248–264. [CrossRef]
18. Anderson, D.P. Boinc: A system for public-resource computing and storage. In Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, USA, 8 November 2004; pp. 4–10.
19. Mawgoud, A.A.; Taha, M.H.N.; Abu-Talleb, A.; Kotb, A. A deep learning based steganography integration framework for ad-hoc cloud computing data security augmentation using the V-BOINC system. *J. Cloud Comput.* **2022**, *11*, 97. [CrossRef]
20. Larson, S.M.; Snow, C.D.; Shirts, M.; Pande, V.S. Folding@ Home and Genome@ Home: Using distributed computing to tackle previously intractable problems in computational biology. *arXiv* **2009**, arXiv:0901.0866.
21. Pandey, A.; Calyam, P.; Debroy, S.; Wang, S.; Alarcon, M.L. VECTrust: Trusted resource allocation in volunteer edge-cloud computing workflows. In Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing, Leicester, UK, 6–9 December 2021; pp. 1–10.
22. Mengistu, T.M.; Albuali, A.; Alahmadi, A.; Che, D. Volunteer cloud as an edge computing enabler. In Proceedings of the Edge Computing–EDGE 2019: Third International Conference, Held as Part of the Services Conference Federation, SCF 2019, San Diego, CA, USA, 25–30 June 2019; Proceedings 3; Springer: Cham, Switzerland, 2019; pp. 76–84.
23. Mounesan, M.; Lemus, M.; Yeddulapalli, H.; Calyam, P.; Debroy, S. Reinforcement Learning-driven Data-intensive Workflow Scheduling for Volunteer Edge-Cloud. *arXiv* **2024**, arXiv:2407.01428.
24. Alarcon, M.L.; Nguyen, M.; Pandey, A.; Debroyand, S.; Calyam, P. VECFlex: Reconfigurability and Scalability for Trustworthy Volunteer Edge-Cloud supporting Data-intensive Scientific Computing. In Proceedings of the 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC), Vancouver, WA, USA, 6–9 December 2022; pp. 151–156.
25. Kantardjian, M.; Elsayed, S.A.; Hassanein, H.S. Dynamic Worker Availability Prediction at the Extreme Edge. In Proceedings of the GLOBECOM 2023—2023 IEEE Global Communications Conference, Kuala Lumpur, Malaysia, 4–8 December 2023; pp. 3603–3608.
26. Tighe, M.; Keller, G.; Bauer, M.; Lutfiyya, H. DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management. In Proceedings of the 2012 8th International Conference on Network and Service Management (CNSM) and 2012 Workshop on Systems Virtualiztion Management (SVM), Las Vegas, NV, USA, 22–26 October 2012; pp. 385–392.
27. Markus, A.; Al-Haboobi, A.; Kecskemeti, G.; Kertesz, A. Simulating IoT Workflows in DISSECT-CF-Fog. *Sensors* **2023**, *23*, 1294. [CrossRef]
28. Souza, P.S.; Ferreto, T.; Calheiros, R.N. Edgesimpy: Python-based modeling and simulation of edge computing resource management policies. *Future Gener. Comput. Syst.* **2023**, *148*, 446–459. [CrossRef]

29. Zhang, R.; Chu, X.; Ma, R.; Zhang, M.; Lin, L.; Gao, H.; Guan, H. OSTTD: Offloading of splittable tasks with topological dependence in multi-tier computing networks. *IEEE J. Sel. Areas Commun.* **2022**, *41*, 555–568. [CrossRef]
30. Alonso-Monsalve, S.; García-Carballeira, F.; Calderón, A. Combos: A complete simulator of volunteer computing and desktop grids. *Simul. Model. Pract. Theory* **2017**, *77*, 197–211. [CrossRef]
31. Sebastio, S.; Amoretti, M.; Lafuente, A.L. AVOCLOUDY: A simulator of volunteer clouds. *Softw. Pract. Exp.* **2016**, *46*, 3–30. [CrossRef]
32. Monasterio, V.; Castro-Mur, J.; Carro, J. DENIS: Solving cardiac electrophysiological simulations with volunteer computing. *PLoS ONE* **2018**, *13*, e0205568. [CrossRef] [PubMed]
33. Teacy, W.L.; Patel, J.; Jennings, N.R.; Luck, M. Travos: Trust and reputation in the context of inaccurate information sources. *Auton. Agents Multi-Agent Syst.* **2006**, *12*, 183–198. [CrossRef]
34. da Rosa, J.H.; Barbosa, J.L.; Ribeiro, G.D. ORACON: An adaptive model for context prediction. *Expert Syst. Appl.* **2016**, *45*, 56–70. [CrossRef]
35. Sadana, U.; Chenreddy, A.; Delage, E.; Forel, A.; Frejinger, E.; Vidal, T. A survey of contextual optimization methods for decision-making under uncertainty. *Eur. J. Oper. Res.* **2024**, *320*, 271–289. [CrossRef]