Computer Communications 216 (2024) 95–111

Contents lists available at ScienceDirect

Computer Communications

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

Enhancing modular application placement in a hierarchical fog computing: A latency and communication cost-sensitive approach

Leonan T. Oliveira [a,b,*], Luiz F. Bittencourt [c], Thiago A.L. Genez [d], Eyal de Lara [e], Maycon L.M. Peixoto [a]

[a] Federal University of Bahia - Institute of Computing, Salvador/BA, Brazil
[b] Federal Institute of the North of Minas Gerais, Almenara/MG, Brazil
[c] Universidade Estadual de Campinas (UNICAMP) - Instituto de Computação, Av. Albert Einstein, 1251, Campinas/SP, Brazil
[d] European Bioinformatics Institute, Wellcome Genome Campus, Hinxton, Cambridgeshire, CB10 1SD, UK
[e] Department of Computer Science - University of Toronto, Toronto, Canada

ARTICLE INFO

Keywords:
Cloud computing
Fog computing
Modular application
Resource allocation
Quality of service

ABSTRACT

Over the years, cloud computing has been a key enabler for handling complex applications and services for Internet of Things (IoT) devices situated at the edge of the network. Services and applications that are driven by the IoT environment commonly have stringent latency-sensitive requirements and may experience long network delays due to the long physical distance of cloud-based computational resources from IoT devices. Fog computing gained adoption as a solution in this case because it shortens this distance by spreading the computing power around the edge of the network in tiers. This contributes to network latency reduction and response times improvements of applications that have sensitive temporal requirements, besides improving the overall data traffic management in the network. Nevertheless, when certain requirements of an application are prioritized over others during the resource allocation process, a fog tier closer to IoT devices may experience resource depletion, forcing other latency-sensitive applications to use resources from a distant fog level and causing them to become non-responsive. To address this issue, this work proposes an approach for allocating modular applications in a hierarchical tier-based fog computing architecture. The proposed approach, named **Least Impact - X (LI-X)**, aims to minimize the response time of latency-sensitive applications and reduce data traffic on the network by mitigating the idle time of resources at the lower levels of the hierarchical fog. This is achieved by distributing the application modules among the fog tiers in order to minimize the response time of delay-sensitive applications, while also reducing the overall network traffic. The performance of LI-X was compared to previous studies in a simulated iFogSim environment. Results have demonstrated that LI-X outperforms these studies in most of the proposed scenarios, effectively reducing response time and minimizing communication data costs on the network.

1. Introduction

The recent years have been marked by a substantial growth of the Internet of Things (IoT), characterized by the development of various applications that seek to connect things to the Internet. IoT can serve a wide range of purposes, including improving traffic flow and public transportation, smart grids, and enabling residential automation for tasks such as lighting control, security management, temperature regulation, and even plant watering schedules. IoT can also be leveraged for complex video and audio processing applications such as online gaming and interactive services [1].

IoT devices are equipped with sensors and actuators to collect real-time information and interact with the real world. As a consequence, IoT applications have to process large amounts of data to meet different application requirements efficiently [2–4]. Nevertheless, these devices frequently lack sufficient computational capabilities to process a huge amount of data and make faster decisions. A solution to bring computing capabilities closer to these devices is the employment of cloud computing services by IoT applications. It provides storage and processing resources for the collected IoT data to be able to be executed by complex algorithms and data analysis [5–8].

* Corresponding author.
E-mail addresses: leonan.teixeira@gmail.com (L.T. Oliveira), bit@ic.unicamp.br (L.F. Bittencourt), thiagogenez@ebi.ac.uk (T.A.L. Genez), delara@cs.toronto.edu (E. de Lara), maycon.leone@ufba.br (M.L.M. Peixoto).

https://doi.org/10.1016/j.comcom.2024.01.002
Received 28 July 2023; Received in revised form 24 November 2023; Accepted 3 January 2024
Available online 9 January 2024

In this context, cloud computing acts as a central point of data storage and processing power. It allows resources to be resized according to demand due to its computational capabilities and elasticity features. The drawback in this case falls in the long geographical and logical distances that data needs to travel between cloud data centers and IoT devices. Latency in the communication between these entities can be introduced, which can be a concern for latency-sensitive applications that require real- or near-real-processing time. Even small delays may not be acceptable [9]. In some scenarios [10–13], it may be necessary for data to be transmitted back and forward to IoT devices, making communication delays even longer.

Fog computing arrived to bring cloud computing capabilities closer to IoT devices, mitigating the latency issue associated with data transfers. It aims to reduce the time required for data to travel on the communication channel between IoT devices and computing resources, as well as decreasing overall network traffic as a result of reducing the network distance that data has to travel. To efficiently deliver this computing power widely to IoT devices at the edge of the network, fog computing nodes must be strategically geographically distributed to ensure close proximity to mobile devices in order to minimize data transmission latency and network traffic [14–16]. By allowing data to be processed and stored closer to its source, within trusted administrative domains, fog computing inherently limits the exposure of sensitive data. This localized handling of data not only reduces the vulnerability associated with wide-area network transmissions but also aligns with privacy-preserving practices by confining data within defined, reliable boundaries [17].

Hierarchical fog computing is a distributed computing architecture that organizes and distributes computational resources across multiple levels in a fog computing network. This architecture enables resources to be distributed across different geographic locations, forming a hierarchical structure with interconnected levels. The resources at each level can vary in computational power and connectivity, allowing for a more efficient distribution of resources and improved scalability. Hierarchical fog computing is designed to meet application requirements more effectively than centralized cloud computing, which aggregates resources in a single level or physical location. This architecture provides a more flexible and scalable approach to data processing, allowing for efficient utilization of resources and the ability to quickly adapt to changing application needs [18–21].

Applications have varying resource requirements, such as network utilization and response time. Additionally, these applications can be modular, allowing for the execution of individual modules on different devices at different levels of a hierarchical fog. Given the limited resources and distributed nature of fog computing, service providers must employ application placement algorithms to enhance the execution of these applications and modules to improve resource utilization and satisfy the unique requirements of each application [19,22].

To address the requirements of efficient application placement in a hierarchical fog computing architecture, we propose an approach designed to enhance both the latency and communication cost. This problem involves strategically allocating IoT applications across a distributed fog computing network, which is inherently challenged by limited resources, varying latency requirements, and complex network traffic dynamics. The essence of this problem lies in determining where and how to place different application components within the hierarchical structure of fog computing to optimize performance. This optimization needs to address not only the computational efficiency of individual applications but also the overall network efficiency and resource utilization. Such optimization is crucial for meeting the diverse and dynamic demands of IoT applications, which range from latency-sensitive tasks to large-scale data processing. The goal is to achieve an optimal balance between resource availability, application requirements, and network constraints, thereby enhancing the efficiency and responsiveness of IoT systems in a fog computing environment [14,21,23].

The challenges in optimizing application placement in fog computing are multifaceted and complex. Firstly, there is the issue of accurately assessing and predicting the varying resource demands of diverse IoT applications. This requires a deep understanding of each application's unique characteristics, such as their computational intensity, data size, and latency sensitivity. Secondly, the inherent limitations in fog computing resources, including computational power, storage capacity, and network bandwidth, pose significant constraints. Another major challenge is dynamically adapting to fluctuating network conditions and application demands in real-time, ensuring optimal placement decisions are continually maintained. Additionally, ensuring efficient data transmission and minimizing latency, especially for real-time and critical IoT applications, is a key challenge. These issues are compounded by the need to maintain overall system balance, avoiding resource overloads or underutilization, which can lead to reduced system efficiency and performance degradation.

To address these gaps, our research introduces the **Least Impact - X (LI-X)** approach, a novel strategy designed to enhance both latency management and communication cost efficiency in hierarchical fog computing. LI-X optimally distributes application modules across the computational resources of the fog network, minimizing response time for latency-sensitive applications and reducing overall network traffic. Furthermore, it uniquely considers the communication patterns between application modules, ensuring efficient resource utilization while minimizing the impact on other applications. This approach represents a significant contribution to the field of IoT and fog computing, offering a more effective solution to the pressing challenges of placement applications and latency management in today's increasingly complex IoT environments.

The performance of the proposed approach was evaluated through simulations and experimental evaluations using the iFogSim simulator. The results of this study are expected to contribute to the advancement of the field of fog computing by providing a solution for the efficient placement of modular applications within hierarchical fog computing environments. The contributions of this work are three-fold:

(i) Development of an algorithm for modular applications placement in a hierarchical fog computing environment, incorporating latency and communication costs requirements;
(ii) Improvement of the iFogSim simulator to enable the construction of fog computing topologies in a declarative manner, allowing a more intuitive and efficient design and configuration of hierarchical fog computing environments; and
(iii) Conducting an extensive evaluation of LI-X compared to previous algorithms across different topologies using the iFogSim simulation tool, providing insights on the performance and effectiveness of these algorithms in different scenarios and environments and identifying potential areas for improvement.

The rest of this paper is organized as follows. We present the problem statement in Section 2. In Section 3, we propose the latency and Communication Cost-Sensitive Approach, while in Section 4, we present and evaluate the results. In Section 5, we give an overview of the related work. Finally, we present the conclusion and future work in Section 6.

2. System model and problem statement

The increasing demand for latency-sensitive applications has driven the growth of fog computing, a distributed computing paradigm that extends cloud computing services to the network's edge. To efficiently use computational resources in fog, designing an appropriate hierarchical architecture that considers the diversity of devices and their computing capabilities is crucial. However, a major challenge in Fog Computing is the efficient allocation of resources in a heterogeneous environment, which includes devices with different processing power and connectivity and diverse applications with varying requirements.

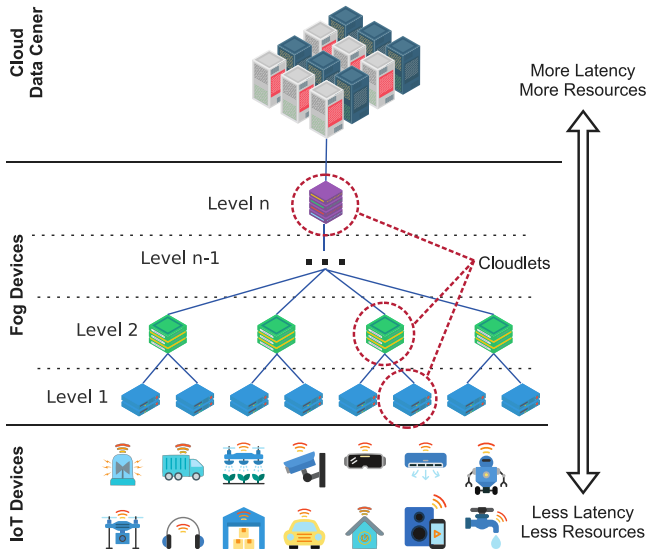


Fig. 1. Overview of a hierarchical fog computing.

Therefore, developing resource allocation mechanisms that efficiently manage resources in such a dynamic and diverse environment is essential.

2.1. Hierarchical fog computing

Fog computing architecture models typically have a three-layer structure, with the cloud at the top, followed by the fog layer, and finally, the IoT (or edge) devices at the bottom layer. The fog layer aims to minimize the latency between IoT devices and computational resources. This is accomplished by utilizing geographically distributed resources near the IoT devices, ensuring a wider coverage area [24,25]. Therefore, the fog layer resources can be organized as a multi-level structure, building a hierarchical organization. Generally, it starts from levels with limited resources in terms of number or capacity, located closer to the IoT devices, then on to more resourceful fog levels that are further away from IoT devices but not as far away as the cloud. Fig. 1 shows that the fog layer can comprise many levels [22,25].

At each level, resources can be distributed among heterogeneous cloudlets, known as micro data centers, nano data centers, or local clouds. In each cloudlet, different resources may be available such as computing power (CPU), RAM, storage, graphic processing (GPU), and communication (network) [19,22,25]. In this work, we assume the level closest to the devices as level 1, the level closest to the cloud as level n, and the intermediate levels are represented by the set of $Levels = \{2, 3, \ldots, n-1\}$. Therefore, it is necessary to choose at which fog level each application module should be executed to meet the requirements of each application.

2.2. Applications

Latency-sensitive applications are benefited from the use of fog computing since the distance necessary to travel the data between the device and a cloudlet tends to be shorter than the distance necessary to travel the same data set between the device and the cloud. Thus it is possible to reduce the total time between the request and the arrival of the response. Fog computing also offers the benefit of reducing data traffic on the network. By bringing computation and storage closer to the edge devices, data has to travel shorter distances, resulting in lower overall network usage and benefiting applications with high data transmission rates, where fog computing can alleviate data flow and reduce transmission costs.

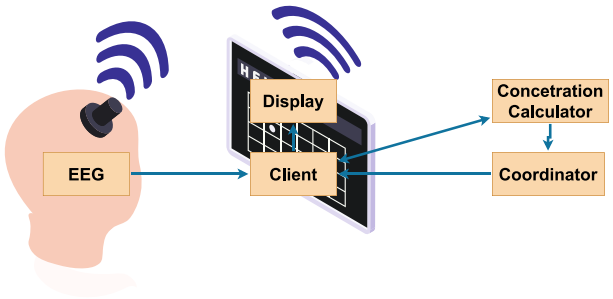


Fig. 2. EEGTBG application modules.

An IoT application operates as a Software-as-a-Service (SaaS), utilizing an Application Program Interface (API) to extract and analyze sensor data. Key components of these applications include sensors and context awareness. Sensors are typically characterized by their compact size, resource limitations, and power constraints. Virtualized servers play a crucial role in processing the generated data. In a broader context, IoT applications can typically be divided into finite modules, facilitating deployment across various physical or logical computing devices. This modular structure enables their deployment across a range of physical or logical computing devices. [21].

In this paper, we consider the applications composed of modular parts, meaning the application comprises modules that can be executed individually in different cloudlets. Moreover, each application can have different priority levels regarding requirements, and each module can use different computational resources. The modules of the applications are allocated to available resources in a fog computing architecture to reduce the delay in delay-sensitive applications and also aim to reduce the data traffic in the network. We assume two different applications, one delay-sensitive and the other that generates high data traffic but does not require a very tight response time. The following sections will present details of each one. The general idea of this work is similar to proposals from three other works in the literature [19,22,26].

2.2.1. EEG Tractor Beam Game – EEGTBG

An electroencephalography Tractor Beam Game (EEGTBG) is a multiplayer game based on Brain–Computer Interface — (BCI), where the user uses their brain to interact with the game with the support of EEG readers. Additionally, it is a latency-critical application, i.e., it requires low delay limits close to real-time. Players must use an EEG headset connected to their smartphones (Fig. 2). The goal is to collect items using concentration; the more concentrated the player is, the more they will attract the items to them [19,26–28].

EEGTBG game has three main processing modules: *Client*, *Concentration Calculator*, and the *Coordinator*. The dependencies between the modules were modeled using definitions of periodicity, selectivity, and data size. For example, a 1 kB tuple is sent from the *concentration calculator* to the *connector* every 100 ms, or for every tuple received by the Concentration Calculator, a 0.5 kB tuple is sent to the Client. Fig. 3 presents a graph with the representation of the modules, and the dependencies between them, which were also used in the works of [19,22,26].

2.2.2. Video Surveillance Object Tracking – (VSOT)

Video surveillance systems are traditionally designed to store data streams humans can later analyze. However, analyzing hours of video footage from multiple cameras is not a trivial task. Recent years have seen the introduction of smart video monitoring systems capable of detecting and tracking objects, detecting actions, and performing scene descriptions, among other features. These systems can reduce or even eliminate the need for human video analysis. On the other hand, video streams can generate a high amount of data traffic, and using only

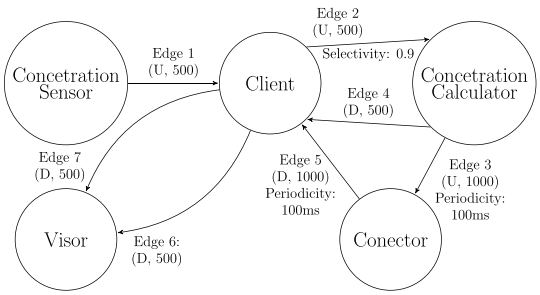


Fig. 3. Modules of *EEG Tractor Beam Game*. *Source:* Adapted from [28].

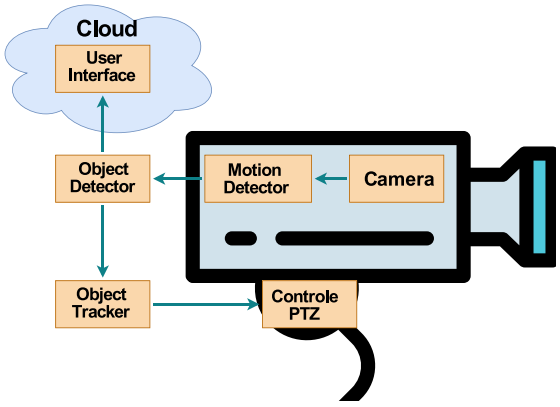


Fig. 4. VSOT application modules.

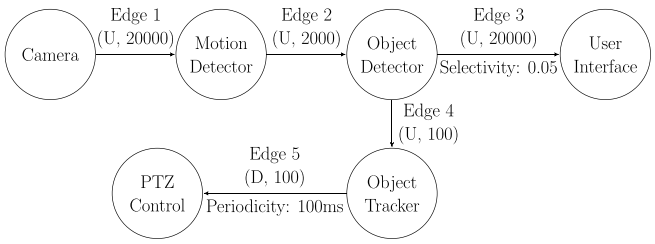


Fig. 5. Modules of VSOT. *Source:* Adapted from [28].

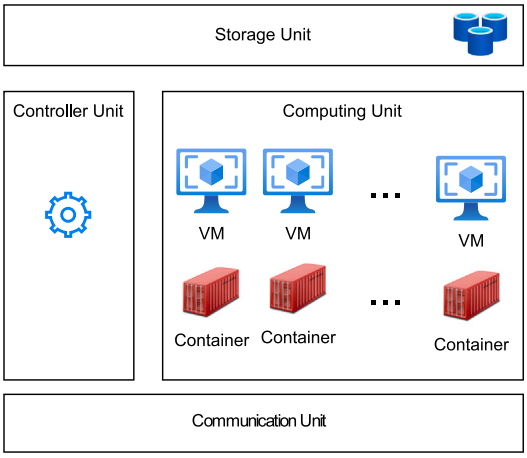


Fig. 6. Basic modules of a *cloudlet*.

cloud resources to process the data may increase network overload and transportation costs [28–30].

Gupta et al. (2017) [28] present a component-based video surveillance system, as illustrated in Fig. 4. The proposed system consists of sensors, actuators, and five modules that perform the tasks of detecting and tracking the movement of objects, as well as providing a user interface: *Motion Detector*, *Object Detector*, *Object Tracker*, *PTZ Control*, and *User Interface*.

Motion Detector module operates internally in the cameras and detects any movement, sending the video stream to the *Object Detector* for analysis. The *Object Detector* module receives the video, compares it with previous images, and checks for movements of a specific object. If detected, it forwards the request to the *Object Tracker* module, which is responsible for calculating the coordinates of the object, the direction of movement, and the new camera position so it can continue to track the object. After making the necessary calculations, the new camera position is sent to the *PTZ Control* module, which operates internally in the camera and will activate actuators to reposition the camera with the new parameters. The *User Interface* module receives relevant video segments identified by the system and provides an access interface for their users. Fig. 5 presents a graph with the representation of the modules, and the dependencies between them, which were also used in the works of [19,22,26].

To efficiently utilize these resources, as each cloudlet has limited resources, applications need to be forwarded to be executed at other levels of the fog architecture. In such situations, decision-making involves choosing which application or application module should be executed on another cloudlet. These resource allocation decisions can be performed based on resource characteristics, application performance requirements, and/or the availability of resources at different levels of the hierarchical fog [19,22].

2.3. Resource allocation

Considering the architecture presented in Fig. 1, the fog layer is composed of different *cloudlets* distributed across various levels. Each cloudlet of a given level l connects with another *cloudlet* from the immediately above level $l+1$ until its final level n (cloud). Based on this premise, we defined a cloudlet as a set of four basic modules (Fig. 6): (i) the storage unit, responsible for providing data persistence for the applications; (ii) the communication unit, responsible for establishing and managing connections with other devices; (iii) the computational unit, responsible for providing resources such as memory and processing power for the application modules, represented in the figure by containers or virtual machines; (iv) and finally the controller unit, responsible for managing the applications and resource utilization.

As limited resources are offered, effective resource allocation is crucial to meet applications' various Quality of Service (QoS) requirements. Resource allocation determines where and with what resources each application will be executed. These decisions directly affect the response time of applications [24,31]. To efficiently utilize these limited resources, allocation algorithms may consider computational processing power, network communication, energy consumption, and storage capacity as inputs [24,31,32].

2.4. Distributed management

Allocation algorithms may also consider the hierarchical layers and levels with its heterogeneous *cloudlets*, and the possibility of sharing resources among themselves [19,28]. We use a distributed approach for deciding where each application module should be executed. Each cloudlet has the autonomy to decide, through its controller unit, which modules of newly arrived applications should be executed and, in some cases, forward them to the next cloudlet in the immediately higher

level. In the context of distributed management, the following key aspects highlight the advantages of this approach:

(i) Reducing Latency: For those fog applications that often have stringent latency requirements, a distributed computing framework helps reduce network latency by bringing computational resources closer to the edge of the network.

(ii) Improving Data Traffic Management: Distributed computing resources across different geographical locations and levels in a hierarchical fog computing architecture contribute to more efficient data traffic management. This leads to a reduction in network congestion and more effective use of resources.

(iii) Enhanced Scalability and Flexibility: A distributed hierarchical fog computing architecture provides greater scalability and flexibility compared to traditional centralized cloud computing models, allowing for dynamic resource allocation based on changing application needs and network conditions.

(iv) Minimizing Communication Costs: The proposed approach, named Least Impact - X (LI-X), minimizes response time for latency-sensitive applications and reduces network data traffic by optimizing the placement of application modules across the fog tiers, thereby reducing communication costs and enhancing system performance.

(v) Adaptability to Changing Application Needs: The distributed nature of fog computing allows for the flexible allocation of modular applications, adapting to changing requirements in IoT environments.

(vi) Minimizing Idle Time of Resources: LI-X approach also minimizes the idle time of resources in lower levels of the hierarchical fog by efficiently distributing application modules, which is a key aspect of effective distributed management.

A hierarchical architecture provides one approach to modeling the edge-cloud continuum, and alternative topologies should be evaluated based on factors like deployment costs and the business models of telecom and edge providers. Take, for instance, a mesh topology that permits horizontal communications; while each configuration involves a cost-performance trade-off, a mesh topology may incur additional infrastructure costs. However, it can enhance the efficiency of horizontal resource allocation. This introduces a larger search space for optimization problems, necessitating more sophisticated and costly resource allocation algorithms.

In the context of upward and downward offloading, the decision-making process assumes proper capacity planning, ensuring sufficient resource capacity at each hierarchical level. Additionally, even within a hierarchical architecture, indirect horizontal communication is conceivable. However, this comes at the cost of increased communication expenses, as more links would be utilized for communication both upward and downward in the hierarchy. It is crucial to recognize that the intricacies of these choices depend on careful consideration of various factors to strike a balance between efficiency and affordability in the deployment of edge-cloud architectures.

The subsequent section will delve into the algorithm proposed in this work, aiming to organize the modules of modular applications in an architecture where their communication is structured vertically. This approach focuses on enhancing communication flows within the architecture, with a specific emphasis on reducing the delay time of sensitive applications while concurrently attempting to minimize data traffic.

3. Hierarchical allocation algorithms

Let $A = \{a_1, a_2, \dots, |A|\}$ be the set of applications and $D = \{d_1, d_2, \dots, |D|\}$ be the set of devices. Each application a_i is represented by a directed graph $G_{a_i} = (V_{a_i}, E_{a_i})$, where V_{a_i} is the set of the app a_i's modules and E_{a_i} is the set of directed arcs connecting a_i's modules. The application requirement for executing the modules of a_i is represented by R_{a_i}. The execution time for the application a_i is represented by T_{a_i}. The fog hierarchy consists of a set of computing resources at some level $F = \{f_1^1, \dots, f_i^j, \dots, f_n^k\}$, where n is the resource id, k is the fog level id, and $f_i^j.B$ is the associated bandwidth vector. Additionally, the system has a set of computing resources in the cloud, $C = \{c_1, \dots, c_i, \dots, c_m\}$, where m is the resource id. All model parameters are shown in Table 1.

Table 1
Table of the model parameters.

App's parameters	Description
$A = \{a_1, a_2, \dots, \lvert A\rvert\}$	Set of applications deployed by the devices
$G_{a_i} = (V_{a_i}, E_{a_i})$	Graph representing the application a_i
V_{a_i}	Set of the app a_i's modules
E_{a_i}	Set of directed arcs connecting a_i's modules
R_{a_i}	Application requirement for executing the modules of a_i
T_{a_i}	Execution time for the application a_i
Devices parameters	**Description**
$D = \{d_1, d_2, \dots, \lvert D\rvert\}$	Set of devices connected to access points
Fog parameters	**Description**
$F = \{f_1^1, \dots, f_i^j, \dots, f_n^k\}$	Set of computing resources in the fog hierarchy
n	Fog resource id
k	Fog level id
$f_i^j.B$	Associated bandwidth vector
Cloud parameters	**Description**
$C = \{C_1, \dots, C_l, \dots, \lvert C\rvert\}$	Set of computing resources in the cloud

Each resource within the fog and cloud computing environments, represented as $R_{f_i^j}$ and R_{c_i} respectively, is defined by key attributes such as computational power, storage capacity, and available bandwidth. These resources are interconnected through a network of communication links, denoted as L, which facilitate data transfer between devices, fog nodes, and cloud nodes. Each link in this network is characterized by its specific data transfer rate and latency. Within this framework, applications, represented as a_i, generate a series of requests RQ_{a_i}, where each request encompasses particular computational demands, storage needs, and data transfer requirements. These requests are intended to be processed within predefined execution times T_{a_i}, ensuring efficient and timely handling of the computational tasks.

The central problem addressed in this article is the optimization of application placement within the fog and cloud computing environments. Our objective is to develop an approach for placing application requests, aiming to minimize latency while maximizing resource utilization efficiency. This optimization challenge, critical for achieving efficient and responsive computing in fog and cloud environments.

The optimization objective is given as minimizing the overall latency for all applications, represented by the sum of individual latencies for each application, Latency_{a_i}, in addition to a weighted measure of resource utilization across the fog and cloud layers, symbolized as $\text{Utilization}_{R,L}$. The weighting factor, λ, is employed to balance the emphasis between latency and resource utilization. This optimization is bounded by several constraints: the capacities of fog resources, $\text{R}_{f_i^j}$, and cloud resources, R_{c_i}, must not be exceeded; data transfers are required to comply with the characteristics of communication links, particularly regarding data rates and latency; and all application requests are to be processed within their designated execution times, ensuring efficient and timely computational performance.

3.1. LI-X algorithm

LI-X algorithm performs resource allocation of application modules to minimize latency in delay-sensitive applications and alleviate network congestion resulting from data traffic. To achieve this, LI-X calculates $\phi(m)$ (Eq. (1)), which represents the average amount of MIPS (Millions of Instructions per Second) required for executing a given module m of an application without forming request queues. In this equation, β is $\sum_{i=j=l=0}^{i<n;j<k;l<|c|} (f_i^j + C_l)$, representing the number

of resources used to process a given request x that module m has previously processed. The average amount of resources required for each request is calculated by summing the number of resources used by each request divided by the number of observed requests. The amount of resources required for the module is calculated by dividing the average resources used to process each request by the frequency of requests received by module m. This estimates the necessary resources for executing module m in any cloudlet.

$$\phi(m) = \frac{\frac{1}{x}\beta}{freq(m)} \tag{1}$$

Algorithm 1 presents the initialization of LI-X. Firstly, the set of applications that need to be instantiated is extracted from the set of all applications A. In line 2, the applications are sorted according to their priority, with latency-sensitive applications having higher priority than the others. In line 3, the modules of each application are extracted and added to a list, which will later have its elements evaluated individually by the following algorithms in an attempt to allocate resources for each of them in the fog, starting from the current cloudlet c.

Algorithm 1 Initialize (Cloudlet c)

Ensure: Cloudlet c
1: $a \leftarrow Subset\ of\ A\ near\ c$
2: $sort(a)$
3: $modules \leftarrow M(a)$
4: **for** m in $modules$ **do**
5: $allocateOnDevice(c, m)$ (Algorithm 2)
6: **end for**

The resource allocation decision process is detailed in Algorithm 2. When a resource allocation request is received for a module m on a cloudlet c, the algorithm checks whether there are available resources to allocate. If available, the allocation is performed (lines 1–3), and the algorithm is terminated. Otherwise, Algorithm 3 is triggered to return a list M_{up} of modules from the current application already allocated in cloudlet c, including the current module m, that would cause the least communication impact if they were reallocated to higher levels of the fog. Once the return of Algorithm 3 is obtained, if the current module m is the only element in the M_{up} list, it will be forwarded for allocation in the cloudlet immediately above cloudlet c in the fog hierarchy (lines 5–7).

If module m is not in the list, or there are other modules in the M_{up} list, the additional resources required to instantiate module m in the *cloudlet* c are calculated (lines 10–12). First, the difference between the available resources in the *cloudlet* c and the resources required for the execution of module m is checked. Then, the number of resources that would be made available if one instance of each module in the M_{up} list were moved to another fog level or architecture layer is calculated. Based on these two pieces of information, the number of instances of each module that should be moved so that resources for module m can be allocated in the *cloudlet* c is determined. Next, it is checked if there are enough modules in the M_{up} list in the *cloudlet* c to be moved. If there are, the modules are removed from the *cloudlet* c and moved to the *cloudlet* above it. Then, resources for module m are allocated in the current *cloudlet* c. If there are not enough modules to release resources with relocation, module m will be sent to the *cloudlet* above.

After the allocation request of a module in a cloudlet c, Algorithm 3 is responsible for selecting the group of modules that are candidates to be sent to the upper levels of the fog. The first step of this algorithm is to find the possible sets S of modules that could be moved to the level above the fog, and this selection is based on Algorithm 4.

Algorithm 2 allocateOnDevice $(cloudlet, module)$

Ensure: Cloudlet c, Module m
1: $hasSpace \leftarrow r(c) \leq n(m)$
2: **if** $hasSpace = true$ **then**
3: $allocate(c, m)$
4: **else**
5: $M_{up} \leftarrow modulesToUp(m, c)$ (Algorithm 3)
6: $c_{parent} \leftarrow P(c)$
7: **if** $\{modulo\} = M_{up}$ **then**
8: $allocateOnDevice(c_{parent}, m)$
9: **else**
10: $cpuToDeploy \leftarrow r(c) - n(m)$
11: $cpuToRelease \leftarrow n(M_{up})$
12: $totalInstances \leftarrow roundUp(cpuToDeploy \div cpuToRelease)$
13: **if** $countSet(c, M_{up}) \geq totalInstances$ **then**
14: **for all** $moduleToUp\ in\ M_{up}$ **do**
15: **for** $i \leftarrow 0;\ i < count;\ i \leftarrow i + 1$ **do**
16: $deallocate(c, moduleToUp)$
17: $allocateOnDevice(c_{parent}, moduleToUp)$
18: **end for**
19: **end for**
20: $allocateOnDevice(c, m)$
21: **else**
22: $allocateOnDevice(c_{parent}, m)$
23: **end if**
24: **end if**
25: **end if**

Algorithm 3 modulesToUp (module, cloudlet)

Require: Cloudlet c
Ensure: Set of modules M moved upwards (elevated).
1: $S \leftarrow modules\ sets\ in\ c$ (Algorithm 4)
2: **for all** $module\ set M \in S$ **do**
3: $Cost_m \leftarrow 0$
4: **for all** $module\ m \in M$ **do**
5: $Cost_m \leftarrow Cost_m + impact\ of\ M$
6: **end for**
7: **end for**
8: $Select\ M\ whose\ Cost_m\ is\ minimum$

Algorithm 4 GenerateModulesSet (Cloudlet c)

Ensure: Cloudlet c
1: $S \leftarrow \emptyset$
2: **for** $module\ m_s \in C$ **do**
3: **for** $module\ m_t \in c$ **do**
4: **if** $\nexists\ Up\ arc\ from\ m_s\ to\ m_t$ **then**
5: $M \leftarrow \{m_s\}$
6: $S \leftarrow S \cup \{M\}$
7: **end if**
8: **end for**
9: **end for**
10: $S_{old} = \emptyset$
11: **while** $S_{old} \neq S$ **do**
12: $S_{old} = S$
13: **for** $set\ M_s \in S$ **do**
14: **for** $module\ m_s \in M_s$ **do**
15: **for** $m_s\ incoming\ edge\ e$ **do**
16: $M_{m_e} \leftarrow \{m_s\} \cup \{source(e)\}$
17: $S \leftarrow S \cup M_{m_e}$
18: **end for**
19: **end for**
20: **end for**
21: **end while**

For each set, the possible impact caused on the network when the group of modules is moved upwards (or elevated) is calculated. This calculation is performed by Algorithm 5. The group with the smallest impact will be selected as a candidate to be sent to higher levels. Algorithm 4, proposed by [19], generates arrangements of possible modules that can be moved to higher levels of the fog hierarchy. Starting from the analysis of the modules present in a cloudlet c, the algorithm selects and creates an arrangement M of only one element for each module present in cloudlet c that does not initiate communication with other modules already present in the same cloudlet c and adds them to a set of arrangements S (lines 1–9). In the second part of the algorithm, for each arrangement, M present in S, modules responsible for initiating communication with any modules in M are sought. When a module satisfies this criterion, a new arrangement M is created, adding the new module and then adding to S. The process is iterative and repeats until no more new arrangements are created (lines 10–20).

Then, Algorithm 5, proposed in [19], is used to calculate the impact a module m may cause in the network when moved to higher levels of the fog in conjunction with a module set M. To do so, for each edge of the application A to which the module m belongs, Eq. (2) is calculated, where b is the number of bytes transmitted from the source module of the edge to a destination module, s is the selectivity of the edge, T_a is the periodicity at which the application transmits data, and p is the periodicity of the edge. If the edge has m as its source module, the destination module of the edge is not in higher-level cloudlets, and m does not belong to the set M, then the value of Eq. (2) is added to the resulting impact. However, if the destination module is at higher levels, the value of Eq. (2) is subtracted from the resulting impact. On the other hand, if m is the destination device of the edge, the destination module is not found in devices at higher hierarchical levels of the fog, and m does not belong to the set M, the increment is performed. However, the decrement will be performed if the destination module is allocated at higher levels of the hierarchy.

$$I_e \leftarrow b \times s \times \frac{T_a}{p} \tag{2}$$

Algorithm 5 calcImpact (moduleSet M, module m)

Require: *module set* $M; m \in M;$ *Application arcs* E
Ensure: $totalImpact_m$
1: **for all** *application edge* $e \in E$ **do**
2: $I_e \leftarrow b \times s \times \frac{T_a}{p}$
3: **if** m is the source of e **then**
4: **if** No upward device has the target module of e **then**
5: **if** *target module of* $e \notin M$ **then**
6: $totalImpact_m \leftarrow totalImpact + I_e$
7: **end if**
8: **else**
9: $totalImpact_m \leftarrow totalImpact - I_e$
10: **end if**
11: **end if**
12: **if** m is the target of e **then**
13: **if** No upward device has the source module of e **then**
14: **if** *source module of* $e \notin M$ **then**
15: $totalImpact_m \leftarrow totalImpact + I_e$
16: **end if**
17: **else**
18: $totalImpact_m \leftarrow totalImpact - I_e$
19: **end if**
20: **end if**
21: **end for**

LI-X most often uses cloudlets at the lower level of the fog, making application execution tend to be done close to the IoT device when resources are available. This reduces response time and minimizes the data transmitted in the network. Furthermore, when lower levels are overloaded, the approach selects modules to move to higher levels that cause the least impact on the data network.

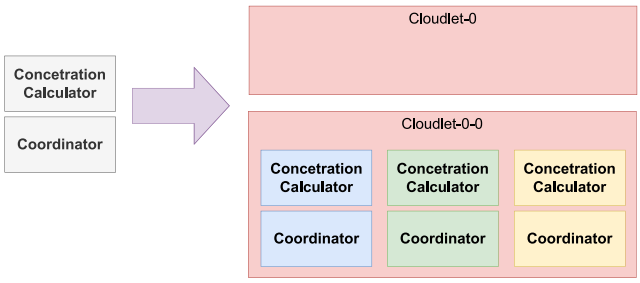


Fig. 7. Allocation request example.

Now, let us delve into the complexity analysis of the LI-X. Assuming that resources need to be allocated to modules (w) of an application, and considering that communication between these modules is represented by a directed graph where the set of edges (e) represents the communication between the modules (w). It is observed that Algorithm 5 has a complexity of $O(w)$ since its execution depends only on the number of edges (e) in the application in question.

Additionally, Algorithm 4 is responsible for generating possible combinations of modules that could have their resources allocated together, taking into account their communication. This algorithm has an approximate complexity of $O(2^w)$ in the worst case, which occurs when the application has a complete communication graph, meaning that all modules interact with each other, resulting in a complete graph. Moving forward, the first line of Algorithm 3 executes Algorithm 4 and traverses the sets with the objective of finding the one that has the least impact when the modules are allocated together on the same device. Since it makes use of Algorithm 4, its complexity is also approximately $O(2^w)$.

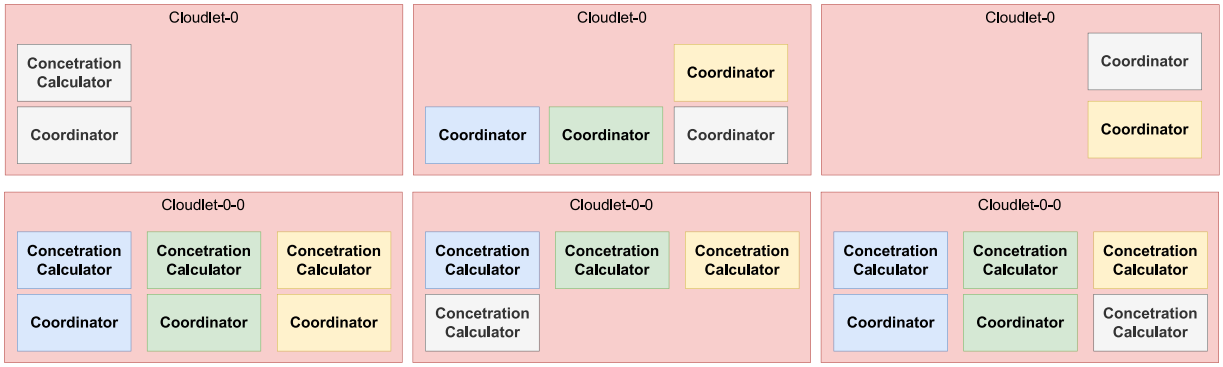
Next, Algorithm 2 comes into play, responsible for allocating resources to modules or forwarding modules to other fog levels. In line 5, it utilizes Algorithm 3, which represents its highest degree, to find the set of modules that could be grouped together and cause the least communication impact on the network, resulting in an approximate complexity of $O(2^w)$. It is worth noting that this complexity depends on the number of modules in the application, and for the considered scenarios, it has a low execution time since both applications used have a small number of modules. The next section presents an illustrated demonstration of what is expected from this proposal and a comparison with other approaches from the literature.

3.2. Toy example

To demonstrate the process of the allocation policy, assume a latency-sensitive application in which the proposed LI-X algorithm is compared to other algorithms from the literature (DP-I and CB-E). This toy application is based on the EEGTBG application (Fig. 3) and has six allocated modules at the lower level cloudlet. Suppose that each module uses 1000 MIPS. As shown in Fig. 7, there are two modules per user, which are grouped by different colors. The *Concentration Calculator* module and the *Coordinator module* from a single user can be allocated at the same time at different levels of the fog hierarchy.

Suppose the cloudlet at the lower level, where the six modules are currently running, has 6000 MIPS of CPU resources and becomes overloaded and cannot accept any new modules. In this scenario, the overloaded cloudlet is identified as *Cloudlet-0-0*, while the upper-level cloudlet *Cloudlet-0* remains unoccupied. Subsequently, a new user enters the system with two modules, represented by white rectangles in Fig. 7. In order to ensure the system's performance and efficiency, the algorithm needs to determine which modules should be relocated to the upper level of the fog hierarchy.

To illustrate the allocation process, we evaluated three different algorithms: DP-I, CB-E, and LI-X. Fig. 8(a) shows the behavior of the DP-I algorithm upon the arrival of new modules. When there is insufficient



(a) DP-I (b) CB-E (c) LI-X

Fig. 8. Outcome example.

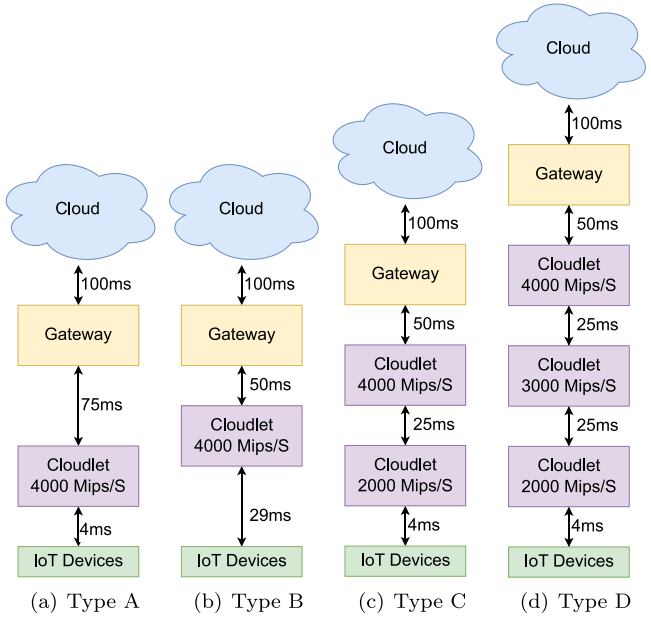
space for allocation at a lower level, DP-I redirects the new modules to higher levels of fog computing, in this case, Cloudlet-0.

The CB-E algorithm, illustrated in Fig. 8(b), considers the communication impact on the executed data traffic before deciding which modules to elevate. Suppose that the algorithm identified all Coordinator modules as the best choice for elevation since it would cause the least communication impact. Therefore, the algorithm elevated all modules of this type to the level above fog computing.

In contrast, Fig. 8(c) shows the result when modules are allocated using the LI-X algorithm. The operation of LI-X is similar to that of CB-E, differing in the way the decision is made about which modules should be moved to higher levels. A series of steps are performed for each module that needs to have resources allocated. Considering that initially, the selected module is the "Coordinator", which will be considered the current module m. The first step is to check if there are enough resources for the new module m in the current cloudlet (Algorithm 2, lines 1–3). Assuming that, in the example, there is no space in Cloudlet-0, line 5 is executed. Algorithm 3, triggered on line 5, originates from the CB-E proposal. Assuming the same scenario, it will return the M_{up} set containing only the "Coordinator" module as a candidate to be moved. Line 7 is then considered true, and the newly arrived m module "Coordinator" is forwarded to Cloudlet-0-0.

In Cloudlet-0-0, the process restarts, but as there is enough space, resources are allocated to the module (Algorithm 2, line 3). Next, the process restarts with the module m being the "Concentration Calculator" that will try to be allocated in Cloudlet-0. As there is not enough space, line 5 is executed. Assuming the same scenario, line 5 of Algorithm 2 is executed, and Algorithm 3 returns the M_{up} set with only the "Coordinator" as a candidate module. However, now line 7 is not true, and the process of moving modules with already allocated resources begins. Line 10 returns the number of resources needed for the "Concentration Calculator" module to be allocated, subtracting the available resources in the cloudlet $r(c)$ from the resources required to run this module $n(m)$. Assuming the values in the example, the $cpuToDeploy$ variable will be −1000, and $cpuToRelease$ is the number of resources that would be freed when raising an instance of each of the modules in the M_{up} set. In this case, the answer will be 1000, as the M_{up} set contains only the "Coordinator" module.

Next, on line 12, it is determined how many instances from the M_{up} set need to be moved for the current module m to have resources allocated in Cloudlet-0. On line 13, the algorithm checks if there are enough instances from the M_{up} set in the current cloudlet that could be moved to make room for the module m. If there are, the resources for each of the instances are deallocated, and these modules are forwarded to Cloudlet-0-0. In our example, one of the "Coordinator" modules would be sufficient. This module is then forwarded to Cloudlet-0-0, and another attempt is made to allocate module m in the current cloudlet (line 20). As there is now available space, the "Concentration Calculator" module will be allocated, resulting in the configuration illustrated in Fig. 8(c).



(a) Type A (b) Type B (c) Type C (d) Type D

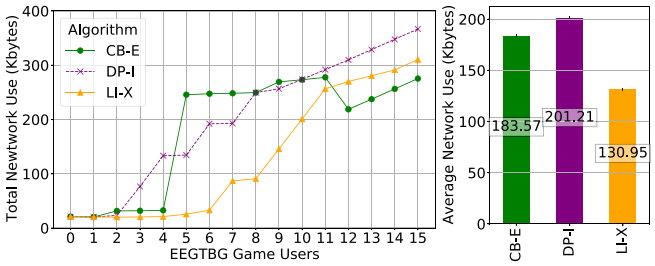
Fig. 9. Fog hierarchical topologies.

The following section will present the experimental environment used to evaluate our proposal.

4. Experiments and results

We defined four different fog topologies (shown in Fig. 9) to conduct our experiments. Each topology has the cloud in the upper layer, followed by a *Gateway* with a delay of 100 ms. These evaluated topologies comprise one or more levels of *cloudlets*, ending with the layer of IoT devices. There are varying levels of communication delay between the different layers. The Type A (Fig. 9(a)), Type B (Fig. 9(b)), and Type C (Fig. 9(c)) topologies are the same as those used in the works of [19,22,26]. We also propose a new Type D topology, a variation with three levels of cloudlets. These different topologies in this work aim to evaluate the performance of different application allocation methods in different scenarios that can frequently occur when a fog computing infrastructure is considered.

For all topologies shown in Fig. 9, 4 instances of VSOT application (Fig. 5) were added in the IoT Devices layer. As for the EEGTBG application (Fig. 3), 16 different scenarios were created to simulate the



(a) Moved Users (b) Average

Fig. 10. Network usage: Topology A.

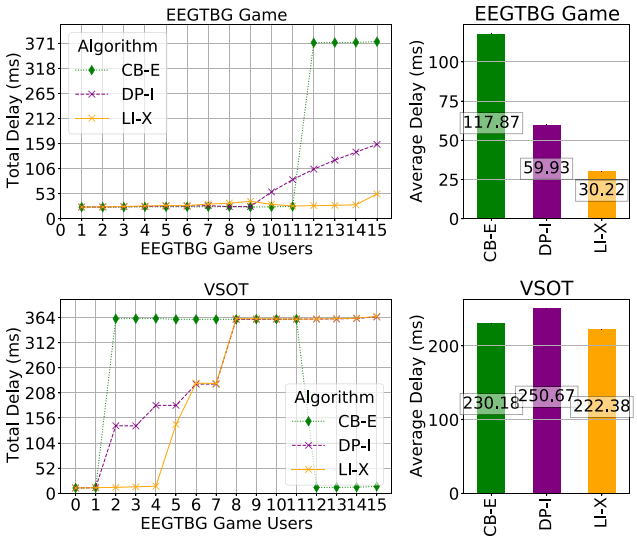


Fig. 11. Delay: Topology A.

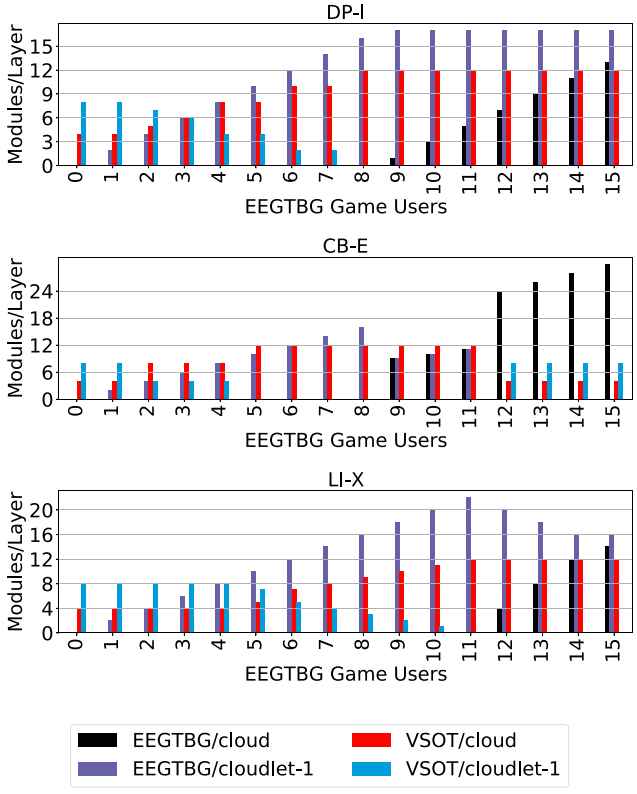


Fig. 12. Module per device: Topology A.

mobility of the game's users. In this sense, scenarios were evaluated, simulating the absence of players until the arrival of 15 EEGTBG players. The Camera, Motion Detector, and PTZ Control modules of the VSOT application are allocated in the Camera itself, and the User Interface module is allocated in the cloud (Fig. 4). The other modules are allocated in the cloud or fog according to the algorithm's decision. The EEG, Display, and Client modules of the EEGTBG application are allocated with the player device. The other modules are allocated in the fog or cloud (Fig. 2).

To serve as a comparative basis for this proposal, we used the DP-I algorithm proposed in [22] and the CB-E algorithm proposed in [19]. The DP-I resource allocation algorithm prioritizes latency-sensitive applications and assigns all modules of these applications to the lowest-level cloudlet for execution. If resources are unavailable at the lowest level, the algorithm allocates modules to higher levels and uses the cloud as a last option for execution [22]. CB-E is an allocation policy that prioritizes latency-sensitive applications at the lower levels of the fog, similar to DP-I. However, when resources become scarce, it selects modules that could be relocated to higher fog layers with minimal impact on network traffic. The algorithm then sends the selected module types to the subsequent levels and layers of the architecture [19]. For each combination of topology (Fig. 9) with each of the algorithms (DP-I, CB-E, and LX-I), ten simulation runs were performed, varying the iFogSim simulator seed for each set.

4.1. Topology A

We analyzed the network traffic of different algorithms in a Topology Type A scenario (Fig. 9(a)). The results showed that LI-X had a slight advantage over CB-E and DP-I regarding average network traffic, with LI-X having an average of 130.95 kB of data traffic while CB-E and DP-I had 183.57 kB and 201.21 kB, respectively. It is worth noting that CB-E presented a gain over LI-X from the arrival of the 12th user, as shown in Fig. 10(a). Nonetheless, LI-X outperformed the compared algorithms in the overall experiment, with a gain of 28.7% over CB-E and 34.9% over DP-I. These findings suggest that LI-X may be a more suitable solution for scenarios similar to the one analyzed in this study.

Regarding the application delay in the Topology A scenario (Fig. 11), it was observed that in the EEGTBG application, LI-X achieved an average delay time of 30.22 ms. In comparison, CB-E had an average delay time of 117.87 ms, and DP-I achieved an average delay time of 26.93 ms. It is possible to notice a similar behavior between the algorithms up to the 9th user, but from the 12th user, the CB-E starts to differ from the other algorithms. When evaluating the behavior of the VSOT application, LI-X also performed better, with an average delay time of 222.38 ms. CB-E presented the second-best result with a delay time of 230.18 ms, and finally, DP-I had an average response time of 250.67 ms. Therefore, LI-X was better than CB-E by 3.39% and DP-I by 11.29%.

In order to understand the reasons for the behavior presented above, Fig. 12 shows the distribution of application modules among devices in Topology A. As shown in Fig. 9, from the arrival of the 12th user, all EEGTBG modules are allocated in the *cloud*, leading to higher delay due to the distance between devices. On the other hand, some modules of the VSOT application were allocated in *Cloudlet-1*, which is closer to the user, reducing the delay and total data traffic over the network for this application. This also explains the behavior identified in the graphs of Fig. 10.

In Fig. 12, DP-I and LI-X exhibit similar behavior, particularly when the number of users is high. Fig. 13 provides a more detailed view of how EEGTBG modules are distributed across devices from the arrival of

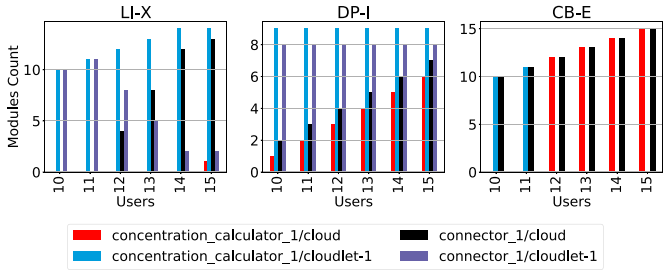
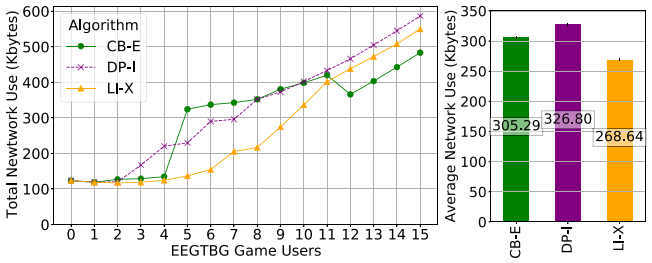


Fig. 13. Modules per device: Topology A. Highlight when the system is overloaded.



(a) Moved Users (b) Average

Fig. 14. Network usage: Topology B.

the 10th user. Highlighting when the 12th user arrives, CB-E allocates all EEGTBG modules in the cloud, while LI-X prefers to allocate a larger number of instances of the *concentration_calculator* module in the Cloudlet-1. This module was identified as a potentially significant cause of communication impact when allocated further away from the user. LI-X allocated the *connector* module as much as possible with the remaining resources. In contrast, the other instances of the modules were moved to the cloud due to insufficient resources in the Cloudlet-1.

4.2. Topology B

Figs. 14(a) and 14(b) present the total and average network traffic, respectively, for Topology Type B (Fig. 9(b)) as users arrive in the system. Despite a smaller difference, LI-X still had the lowest average network usage compared to CB-E (305.29 kB) and DP-I (326.80 kB), with an average of 268.64 kB. As with Topology Type A, CB-E gained an advantage over LI-X from the arrival of the 12th user, as shown in Fig. 14(a). However, in the experiment's average, LI-X had a 12.01% gain over CB-E and a 17.80% gain over DP-I.

The average delay time in Topology B for the EEGTBG application is presented in Fig. 15. LI-X had an average delay time of 79.61 ms, CB-E had an average of 154.22 ms, and DP-I had an average of 104.95 ms. In this scenario, LI-X had a gain of 48.4% over CB-E and 24.2% over DP-I. For the VSOT application, CB-E had an average delay time of 248.93 ms, DP-I had an average of 266.59 ms, and LI-X had an average delay time of 242.45 ms, representing a gain of 2.67% over CB-E and a gain of 9.1% over DP-I.

In Topology B, the distribution of modules among devices is shown in Fig. 16. This topology is similar to Topology A, as both have only one fog level with the same amount of computational resources, differing only by the distance between devices. Due to this, the results regarding the allocation location of each module instance of the application also showed significant similarities. The average delay increases due to the additional distance between the devices and the first fog level. From the arrival of the 9th user, CB-E starts to allocate all instances of the EEGTBG application modules in the cloud, causing a significant increase in the average delay. And from the arrival of the 12th user, CB-E allocates all modules of the VSOT application in Cloudlet-1,

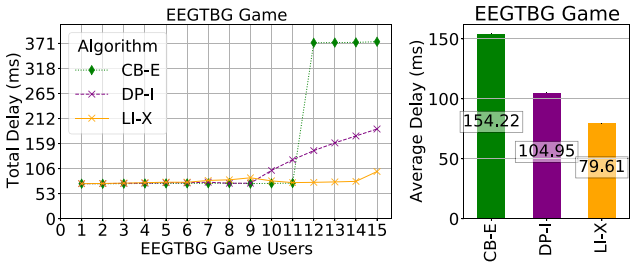


Fig. 15. Delay: Topology B.

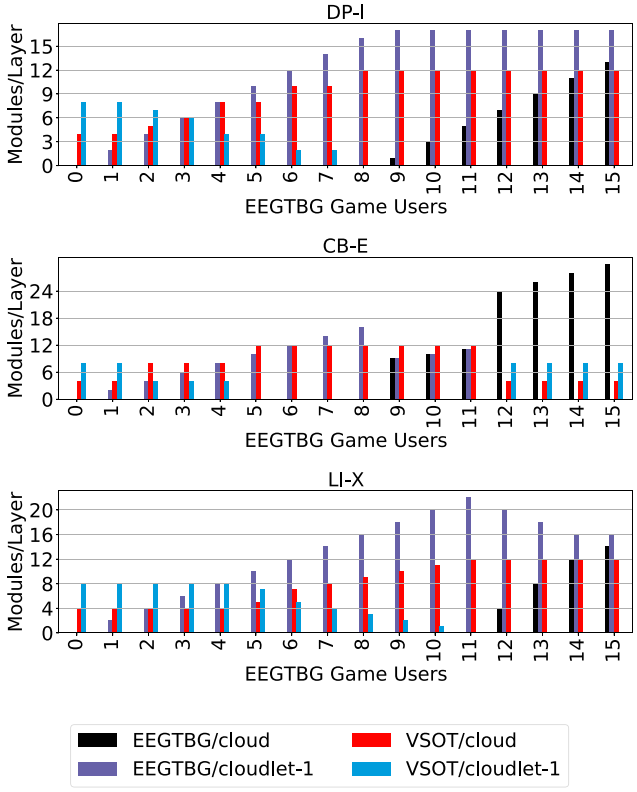


Fig. 16. Modules per device: Topology B.

which ends up reducing the average delay for this application and the total network traffic since VSOT is the application that generates and transmits the most data.

The module distribution of the EEGTBG application instances among the devices in Topology B is presented in detail in the graphs of Fig. 17, starting from the arrival of the 12th user. Once again, it is observed that CB-E allocates all application modules in the cloud, while DP-I distributes the modules proportionally, respecting the resource limit. LI-X prefers allocating the concentration_calculator module in Cloudlet-1, which reduces the average delay time for the EEGTBG

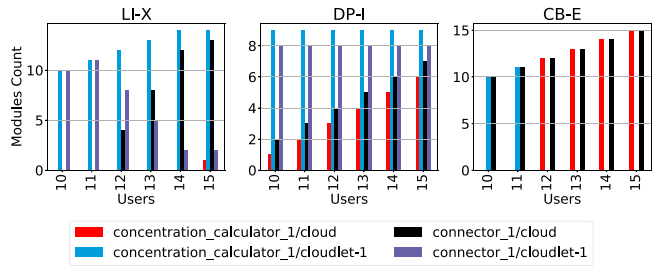
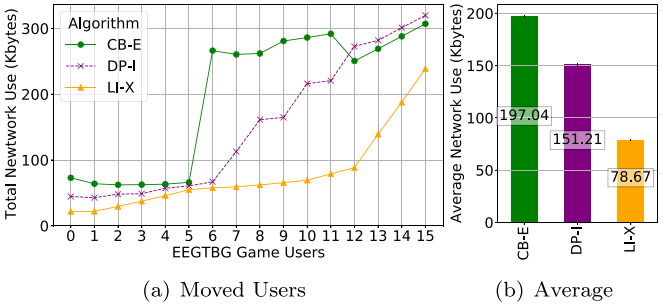


Fig. 17. Module count per device: Topology B. Highlight when the system is overloaded.



(a) Moved Users (b) Average

Fig. 18. Network usage: Topology C.

application, as the concentration_calculator module has a major impact on communication.

4.3. Topology C

The experiment conducted in Topology C involved the addition of a new level of fog. The corresponding network traffic was captured and analyzed for varying user arrival rates. Fig. 18(a) displays the network traffic for each arrival rate, while Fig. 18(b) shows the average network traffic. The performance of three different allocation algorithms, LI-X, CB-E, and DP-I, was compared in this topology. The results indicate that LI-X outperformed the other two algorithms, with an average network traffic of 78.67 kB, compared to CB-E's average of 197.04 kB and DP-I's average of 151.21 kB. This trend was observed for all workloads, as illustrated in Fig. 18(a). On average, LI-X achieved a gain of 60.08% over CB-E and 47.98% over DP-I.

In terms of delay evaluation for Topology C (Fig. 19), the EEGTBG application exhibited an average delay time of 40.27 ms with LI-X, 139.33 ms with CB-E, and 52 ms with DP-I. LI-X outperformed the other algorithms with a gain of 71.1% over CB-E and 22.6% over DP-I. Similarly, for the VSOT application, LI-X performed better in reducing the average delay time, resulting in a 14.1% reduction compared to CB-E and a 30.2% reduction compared to DP-I.

Analyzing the allocation of application modules in Fig. 20, CB-E does not allocate any VSTO application module at the first fog level, which has only half the resources available in Topologies A and B. The same issue occurs with the EEGTBG application after the arrival of the 6th user. This is due to CB-E's decision to move all module instances to the upper level when there is insufficient space to allocate them all, leaving the first level of fog unused. This behavior also extends to upper levels as more users join the system. By the 12th user, the EEGTBG application is entirely allocated in the cloud, increasing the average delay.

Fig. 20 shows that DP-I and LI-X can utilize all fog levels as EEGTBG users arrive in the system. DP-I and LI-X allocate application modules from lower levels up to the cloud when resources are scarce in the lower levels. However, the two approaches differ in how they distribute the application modules. In the distribution of EEGTBG application

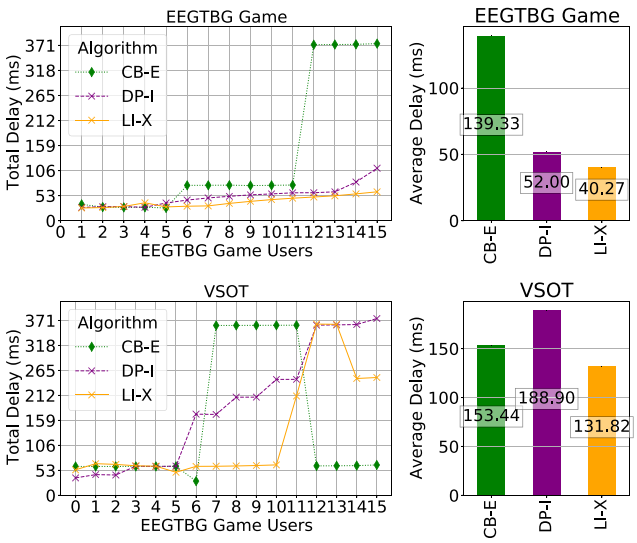


Fig. 19. Delay: Topology C.

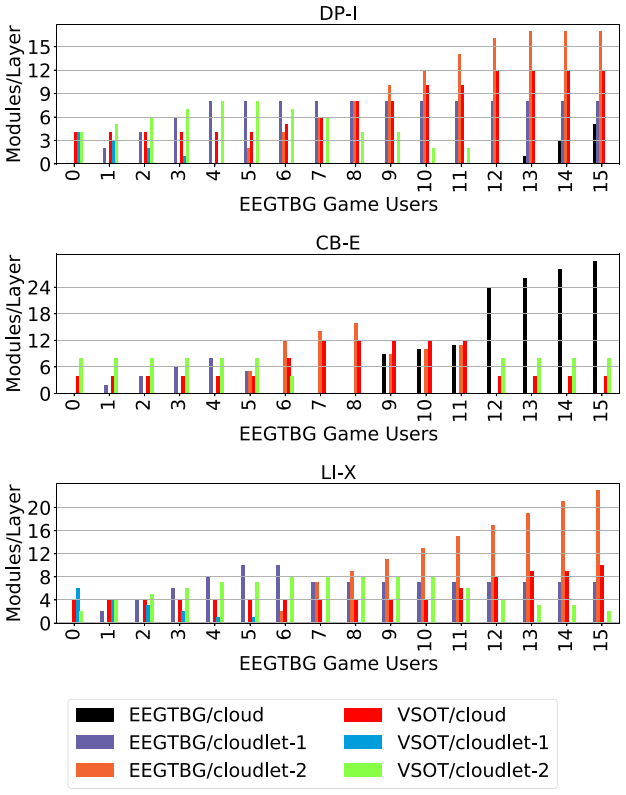


Fig. 20. Modules per device: Topology C.

modules from the 10th user's arrival (Fig. 21), DP-I distributes the modules more evenly, allocating them on the same device whenever it obtains all the application modules for the new user. On the contrary, LI-X prioritizes the allocation of the concentration_calculator module in the lower levels. If it is not feasible to allocate it there, it is allocated in the upper levels. This approach enables LI-X to reduce delay since the concentration_calculator module generates lower data traffic and minimal communication with other application modules, reducing overall data traffic.

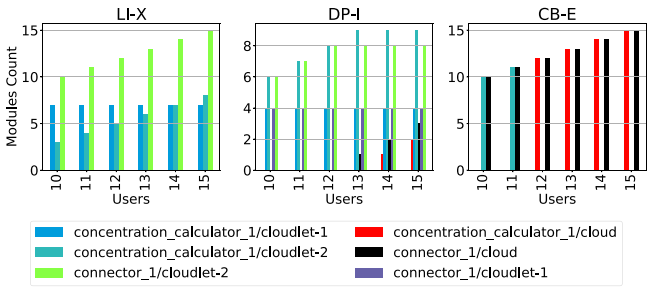
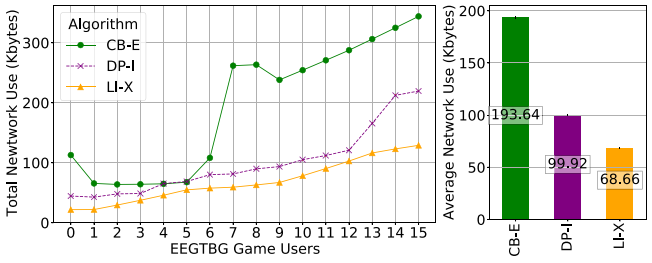


Fig. 21. Module count per device: Topology C. Highlight when the system is overloaded.



(a) Moved Users (b) Average

Fig. 22. Network usage: Topology D.

4.4. Topology D

In Topology D, a third level of fog is added. Fig. 22(a) shows the total network traffic according to the arrival of new users, and Fig. 22(b) presents the average network traffic for the Type D topology (Fig. 9(d)). In this topology, LI-X also achieved a significantly better result than the other algorithms, with LI-X having an average of 68.66 kB, CB-E having an average of 193.64 kB, and DP-I having an average of 99.66 kB. LI-X outperformed the other algorithms for all user quantities in this topology, as shown in Fig. 22(a). On average, LI-X had a gain of 68.55% compared to CB-E and 31.29% compared to DP-I.

After analyzing the data traffic of the network (Fig. 23), LI-X was found to reduce the average delay time by 79.7% compared to CB-E and by 22.9% compared to DP-I for the EEGTBG application. The average delay times were 40.42 ms, 198.93 ms, and 52.42 ms, respectively. For the VSOT application, CB-E had an average delay time of 64.13 ms, DP-I had a delay time of 116.07 ms, and LI-X had a delay time of 81.76 ms. This corresponds to a performance loss of 27.5% compared to CB-E, and a gain of 29.6% compared to DP-I. Although LI-X did not achieve the best time for the VSOT application, it was able to meet the objective of reducing the average delay time of the sensitive application.

Upon evaluating the allocation of application modules across the levels of the fog in Topology D, it is observed from the graphs in Fig. 24 that, even in the absence of EEGTBG users, CB-E fails to allocate any module of the VSOT application at the first fog level. Instead, it distributes the modules of VSOT among the upper levels of the fog and the cloud for almost all user quantities. This behavior is due to the algorithm's inability to allocate instances of the same module individually at different levels. As for the EEGTBG application, all instances begin to be migrated to upper levels according to the available resources as users arrive. This behavior causes resource idleness at lower fog levels and increases the delay in each interaction. As a result, the module instances are allocated further away from the user, as observed in Fig. 23.

DP-I allocates resources based on availability at lower levels, which avoids resource idleness but leads to a lack of resource utilization. However, LI-X considers each module's behavior and allocates them, favoring those that cause less impact. In Fig. 25, the distribution of

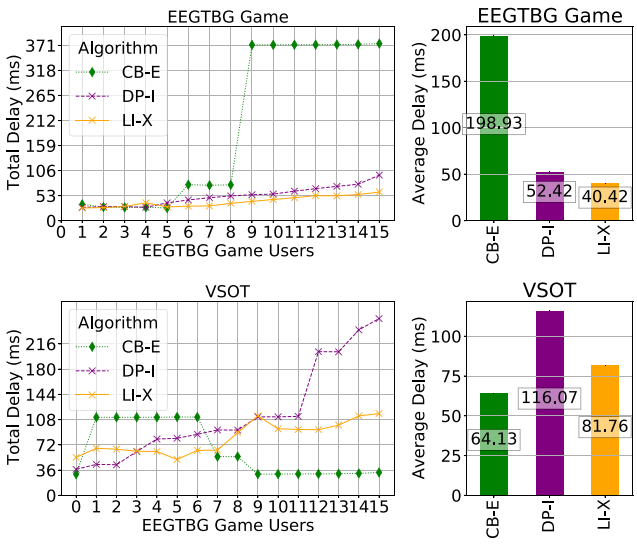


Fig. 23. Delay: Topology D.

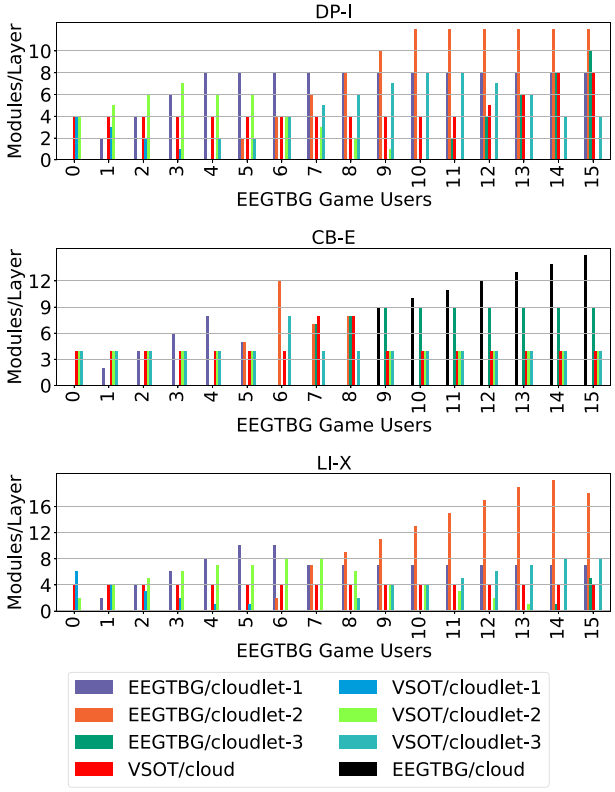


Fig. 24. Modules per device: Topology D.

application modules among the fog levels can be observed, starting from the arrival of the 10th user. The concentration_calculator module is allocated in lower levels, while the connector module is allocated in higher levels according to resource availability. This way, LI-X can reduce the average delay time since the concentration_calculator module is the EEGTBG application module that causes the most significant impact being allocated farther away from the user.

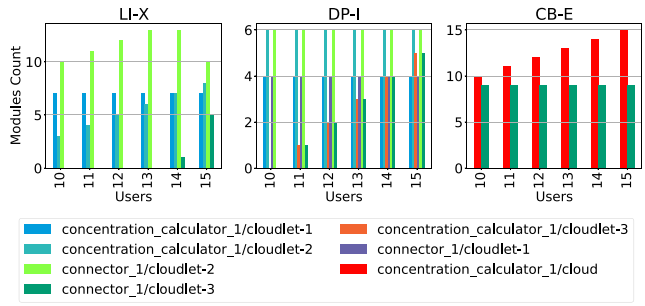


Fig. 25. Module count per device: Topology D. Highlight when the system is overloaded.

Table 2
Network usage (in kB).

Topology	CB-E	DP-I	LI-X	≈CB-E	≈DP-I
A	183.57	201.21	**130.95**	−28.7%	−34.9%
B	305.29	326.80	**268.64**	−12.01%	−17.80%
C	197.04	151.21	**78.67**	−60.08%	−47.98%
D	193.64	99.92	**68.66**	−68.55%	−31.29%

Table 3
Average delay (in milliseconds).

Topology	App	CB-E	DP-I	LI-X	≈CB-E	≈DP-I
A	VSOT	230.18	250.67	**222.38**	−3.39%	−11.29%
	VRGame	117.87	59.93	**30.22**	−74.36%	−49.57%
B	VSOT	248.93	266.59	**242.45**	−2.60%	−9.06%
	VRGame	154.22	104.95	**79.61**	−48.38%	−24.14%
C	VSOT	153.44	188.90	**131.82**	−14.09%	−30.22%
	VRGame	139.33	52.00	**40.27**	−71.10%	−22.56%
D	VSOT	64.13	116.07	**81.76**	+27.49%	−29.56%
	VRGame	198.93	52.42	**40.42**	−79.68%	−22.89%

4.5. Analysis

The network usage of the algorithms used in the study and the difference of LI-X compared to the other algorithms are summarized in Table 2. It was found that CB-E outperformed DP-I when lower-level fog cloudlets had enough resources to allocate all instances of specific module groups. However, when there were not enough resources for all instances, CB-E elevated entire groups of module instances to higher-level cloudlets, increasing the distance to IoT devices and impacting data traffic between modules. On the other hand, LI-X achieved better results for all topologies because it reduces network usage by positioning groups of module instances that have a greater impact on communication within the same cloudlet, a strategy similar to CB-E, and reduces resource idleness at the lower-level fog cloudlets, a strategy similar to DP-I. This behavior was observed in all presented scenarios, showing that LI-X is a good solution for these scenarios.

Table 3 presents detailed information about the delay results for the application requests in all experiment scenarios. As shown, the LI-X algorithm reduced the average delay time for the sensitive application in all cases and also managed to reduce the delay for the non-sensitive application in almost all scenarios, with only a 27.49% loss in topology type D. This was achieved by the strategy of allocating resources to groups of modules with higher communication impact close to each other, which reduced the time required for communication between these groups. Additionally, LI-X reduced the idleness of resources at lower levels of the fog and was more selective when moving groups of modules, placing more related modules in the lower levels of the fog. As a result, more module instances of the applications were executed closer to the IoT devices, thus reducing delays in communication.

In this way, LI-X has shown to be a more efficient solution when compared to the compared works. This conclusion is supported by the results obtained in the experiments, which demonstrated the ability of LI-X to reduce delay, optimize the use of resources, and minimize network traffic in fog computing environments. Moreover, these findings contribute to the advancement of the field and provide valuable insights for developing future hierarchical fog computing systems. It could be a valuable addition to the existing literature on this topic, which we present in the next section.

5. Related work

Due to the various limitations and challenges associated with using cloud computing for certain applications, particularly those that use the Internet of Things (IoT) paradigm, fog computing is consolidating as an option by providing computational power in proximity to the user. Various types of applications can benefit from this paradigm, particularly those that require fast responses, systems that work with facial recognition, voice recognition, and translation, health-related applications, surveillance and emergency systems, and real-time games, among others [33]. However, due to the nature of fog computing and its applications, a series of other challenges have been presented and begun to be studied.

In Shah-Mansouri e Wong (2018) [34], the authors study task allocation in a hierarchical fog computing architecture to maximize user experience by reducing the response time of applications, as well as energy cost reduction. To achieve this, they use algorithms and game-theoretic approaches. According to the authors, the results of the experiments showed that users generally achieved a better Quality of Experience (QoE) compared to other proposals in the literature. However, this work does not address modular applications but rather a task execution system, where applications can be executed at any time on any device as long as the task is directed to that device.

In Yangui et al. (2016) [35], an architecture for a Platform-as-a-Service (PaaS) is proposed to automate the provisioning of applications in hybrid cloud–fog environments. The proposed PaaS architecture allows for the development of applications according to the target domain, configuring and scaling resources to deploy and execute application modules, managing the flow of execution, monitoring Service Level Agreements (SLA), migrating modules, and providing interfaces for resource and module management. Different scenarios of application allocation were observed, and different results were found for each allocation arrangement. The author concludes that the ideal organization of application modules is not a trivial task and requires sophisticated algorithms to achieve better results.

According to Chamola et al. (2017) [33], a Software-Defined Network (SDN) containing multiple cloudlets can be used to execute services in proximity to mobile device users. Executing tasks on a cloudlet network can be a solution for improving QoS in terms of service response time. Based on the proposed policy, if a cloudlet becomes overloaded, tasks should be transferred and processed on another available cloudlet in the network. The management and distribution of tasks are handled by a centralized service called the cloudlet central manager.

Taneja e Davy (2017) [36] presents an algorithm for mapping IoT application modules based on a fog computing architecture that considers the requirements of each application module, such as CPU, memory, and bandwidth. The author aims to allocate the application modules as closely as possible to the users and end devices. To do this, the algorithm considers the resources available in each fog node and seeks to reduce the idleness of resources closest to the network's edge. The author emphasizes the importance of using an architecture based on fog computing and cloud computing when aiming to reduce the response time of applications. Although the algorithm considers that application modules can be allocated on different devices, the authors do not consider the priority of applications or the impact of communication between these modules.

Table 4
Summarizing of related work [19,22,26,33,35–39].

Article	Journal/Congress published	Fog	Hierarchical or multi-level	Requirements		Modular app	Distributed management
				CPU	Network		
Yangui et al. (2016) [35]	IEEE Inter. Symposium on Local and Metropolitan Area Networks	✓	✓			✓	
Chamola et al. (2017) [33]	IEEE Inter. Conference on Pervasive Computing and Communications	✓		✓	✓		
Taneja e Davy (2017) [36]	IFIP/IEEE Symposium on Integrated Network and Service Management	✓		✓			
Aburukba et al. (2020) [37]	Elsevier Future Generation Computer Systems (FGCS)	✓		✓	✓		
Ali et al. (2020) [38]	IEEE Transactions on Cloud Computing	✓		✓			
Kaur et al. (2022) [39]	IEEE Transactions on Services Computing	✓	✓	✓			
Bittencourt et al. (2017) [26]	IEEE Cloud Computing	✓		✓		✓	✓
Charantola et al. (2019) – [DP-I] [22]	IEEE/ACM Inter. Conference on Utility and Cloud Computing	✓		✓		✓	✓
Peixoto et al. (2022) – [CB-E] [19]	IEEE Transactions on Services Computing	✓	✓	✓	✓	✓	✓
This work – [LI-X]		✓	✓	✓	✓	✓	✓

In Aburukba et al. (2020) [37], a customized genetic algorithm is proposed to solve the problem of scaling requests received from IoT devices in a fog computing architecture. With the main goal of minimizing the latency of applications, the proposed approach shows good results compared to other techniques such as round-robin or priority queue. Ali et al. (2020) [38] present an adaptation of the Non-dominated Sorting Genetic Algorithm (NSGA-II), which aims to reduce the response time and execution cost of applications. Both proposals utilize a centralized decision system.

In Kaur et al. (2022) [39], the authors present the Real Time Heterogeneous Hierarchical Scheduling RTH^2S algorithm, designed for task scheduling in real-time in a multilevel fog computing architecture. The premise of the work is that a group of tasks with different sizes, priorities, and deadlines need to be executed. Thus, the algorithm attempts to schedule the execution of the group of tasks considering their different profiles, prioritizing those with higher priorities and shorter deadlines. To evaluate the RTH^2S, the authors used the iFogSim simulation tool and a real-world dataset of task execution records to compare the performance of the proposed algorithm with the direct execution of tasks in a cloud environment, as well as with the Longest Time First (LTF) scheduling algorithm, which attempts to allocate larger tasks to fog nodes where they would be executed more quickly. In their experiments, the RTH^2S achieved better success rates in delivering the most tasks within the deadline and also achieved lower costs for task execution compared to LTF. However, this work does not consider modular applications and resource allocation based on individual tasks.

The work of Bittencourt et al. (2017) [26] discussed the problem of allocating applications on devices in a single-level fog network using the iFogSim simulator. The study analyzed the behavior of two distinct applications in the fog, with different numbers of users moving between regions, taking into consideration three different resource allocation strategies: Concurrent, where the requested applications are allocated concurrently on the same cloudlet without considering resource usage limits, First-Come-First-Served (FCFS), which allocates resources based on the order of allocation requests and directs requests to the cloud when no more resources are available, and finally, Delay-Priority (DP), which takes into account the priority of the application. In this case, applications with tighter response time requirements are prioritized over others. When no more resources are in the cloudlet, the applications are directed to the cloud. As the number of users increased in a particular cloudlet, DP demonstrated better performance in terms of response time for latency-sensitive applications at the cost of increased network traffic. Despite having minimal network traffic, the Concurrent approach did not show good results in terms of latency for the applications since all applications were executed on a single cloudlet concurrently, compromising the execution of the tasks. Bittencourt et al. (2017) [26] did not consider a multi-level fog network scenario and also considered the application modules as a single group that needed to be positioned together on the same device.

In Charantola et al. (2019) [22], the authors propose the Delay-Priority & Individual (DP-I) algorithm for resource allocation in a single-level fog architecture. The study considers that applications can be divided into modules that communicate with each other and can be positioned on different devices. When there is a need to allocate resources for a new application or for new modules of an application already running on a specific cloudlet, and in the absence of sufficient resources, the algorithm, instead of elevating the entire application to the cloud, selects only the dependent modules, which communicate with each other, and directs them to the cloud. Additionally, the algorithm prioritizes keeping applications with lower latency requirements on the cloudlet. The study compares the proposed approach with the algorithms evaluated by Bittencourt et al. (2017) [26]. The impressive results suggest that with the new proposal, it was possible to reduce the average delay in applications with lower latency requirements and that users can experience differences in QoS as services are migrated to the cloud service. However, it does not consider a multi-level fog scenario.

Peixoto et al. (2022) [19] proposes the CB-E (Communication Based & Edgewards), an approach that aims to reduce network traffic while trying to meet the latency requirements of applications. To do this, when there is a need for resource allocation for new applications in a cloudlet, and there are not enough resources, the algorithm evaluates the group of modules that should be migrated. It does this by considering the increase in network traffic that may be produced when certain modules or dependent groups of modules are sent to higher fog levels. Then, it sends to higher levels of the fog all modules of a certain type or groups that cause the least impact on the network. The authors evaluated their results in three different scenarios, considering scenarios with a single-level fog and two levels, also varying the computational capacity of the cloudlets. They compared the proposed approach with the algorithms evaluated by Bittencourt et al. (2017) [26] and the

approach proposed by Charantola et al. (2019) [22]. The effectiveness of CB-E in reducing total data traffic while maintaining acceptable delays in latency-sensitive applications was demonstrated by its superior performance compared to the algorithms proposed in [22,26]. As a result, CB-E proved to be a valuable solution for this scenario.

As seen in Table 4, various studies have been conducted around resource allocation for network architectures based on fog computing applications. Most related works in the review did not consider that the fog layer may have multiple levels. Additionally, only three considered the data traffic generated by communication or the delay caused by the distance between devices. Another important issue is the modular nature of applications, which was only considered in four studies. However, some studies deal only with the execution of isolated tasks and not reserving resources for certain applications. Notably, only three studies use a distributed strategy for decision-making about the location of resource allocation for applications or their modules.

Our study is based on the findings of previous research conducted by other authors in the field of cloud–fog resource allocation. In the first previous one, Bittencourt et al. (2017) [26] present and evaluate the allocation of applications using the FCFS and DP algorithms, which consider the available CPU on the device for the allocation of new applications but do not consider the modularity of the applications. Next, Charantola et al. (2019) [22] present the CB algorithm, which considers the applications' modularity and priorities, aiming to reduce response time. Then, Peixoto et al. (2022) [19] present the CB-E, which in addition to considering the modularity of the applications, also evaluates the impact of communication between each of its modules and presents a two-level hierarchical fog.

Referring to the details presented in Table 4, it provides a comprehensive summary of the model parameters used in the study. It categorizes these parameters into three main groups: App's Parameters, Devices Parameters, and Fog Parameters. Regarding the research gap addressed in our paper: Our study identifies and addresses a significant challenge in the context of fog computing, particularly in the context of applications with stringent latency requirements. The core issue we observed is the application placement problem in fog computing environments. More specifically, when application placement in a fog tier prioritizes certain application requirements over others, it can lead to an imbalance in the system. This imbalance often results in resource depletion in tiers closer to IoT devices. Consequently, latency-sensitive applications are forced to utilize resources from more distant fog levels, leading to increased response times and potential non-responsiveness. To tackle this issue, our paper introduces a novel approach named Least Impact - X (LI-X). This approach is designed to minimize response times for latency-sensitive applications and reduce network data traffic. The key innovation of LI-X lies in its ability to mitigate the idle time of fog resources at lower levels. By strategically distributing application modules across the fog tiers, LI-X aims to balance the need for rapid response times in delay-sensitive applications with the overall reduction of network traffic.

In addition, the proposed LI-X algorithm outperforms previously presented studies, including the CB-E [19], regarding reducing data traffic and meeting application latency requirements. Unlike the CB-E algorithm, which elevates all running instances of a group of modules to higher levels of the fog, the LI-X algorithm selectively elevates only necessary modules to make room for new applications. Furthermore, the LI-X algorithm considers each module's average resource utilization when evaluating resource needs, resulting in reduced resource idleness at lower fog levels. Additionally, this work introduces a fourth hierarchical level in which all algorithms are evaluated.

6. Conclusion

The proposed Least Impact - (LI-X) algorithm builds upon the existing knowledge base by utilizing previous studies' theories, methodologies, and empirical data to contribute to understanding the hierarchical fog topic. To evaluate the performance of the proposed algorithm, LI-X, it was compared to two other algorithms found in the literature, Delay-Priority & Individual (DP-I) and Communication Based & Edgewards (CB-E). For this, a simulated environment was set up using the iFogSim simulator for different fog architecture topologies. Two different applications with different requirements were used, and the response time of the applications in each simulated scenario was evaluated, as well as the amount of data transmitted through the network. At the end of the experiments, LI-X presented better results in all scenarios than the other algorithms.

By referencing the works of other experts in hierarchical fog, a foundation for future research is established. Therefore, to improve the proposed work, the following steps can be taken in the future: (i) Evaluate the impact of live migrations that occur in user mobility scenarios; (ii) Expand and evaluate the scenarios to new types of applications and dynamic topologies of hierarchical fog; (iii) Consider costs and energy efficiency during the resource allocation process; and (iv) Consider applications or tasks that do not require constant resource allocation, such as serverless applications.

In light of the role of data privacy in fog computing, we envision an emerging research avenue in developing privacy-aware allocation algorithms. Such algorithms would strategically place application modules and data in fog nodes, considering privacy requirements alongside computational efficiency. The development of these algorithms presents an interesting challenge, balancing the trade-offs between privacy, latency, and resource utilization. This aspect opens up opportunities for future work where the primary goal is to create sophisticated systems that not only optimize performance but also rigorously protect data privacy in increasingly connected IoT environments.

CRediT authorship contribution statement

Leonan T. Oliveira: Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Luiz F. Bittencourt:** Conceptualization, Formal analysis, Investigation, Project administration, Supervision, Validation, Writing – review & editing, Writing – original draft. **Thiago A.L. Genez:** Conceptualization, Investigation, Project administration, Resources, Validation, Writing – review & editing, Writing – original draft. **Eyal de Lara:** Conceptualization, Data curation, Formal analysis, Investigation, Supervision, Writing – review & editing, Writing – original draft. **Maycon L.M. Peixoto:** Formal analysis, Investigation, Methodology, Project administration, Supervision, Validation, 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

Data will be made available on request.

Acknowledgments

This study was funded in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq, Brazil, FAPESB, Brazil #INCITE-PIE0002/2022, and grant #2019/26702-8, São Paulo Research Foundation (FAPESP), Brazil.

References

[1] Y. Lin, H. Shen, Cloud fog: Towards high quality of experience in cloud gaming, in: 2015 44th International Conference on Parallel Processing, 2015, pp. 500–509.

[2] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing - MCC '12, ACM Press, 2012, p. 13.

[3] M. Chiang, T. Zhang, Fog and IoT: An overview of research opportunities, IEEE Internet Things J. 3 (6) (2016) 854–864.

[4] D.C. Nguyen, M. Ding, P.N. Pathirana, A. Seneviratne, J. Li, H. Vincent Poor, Federated learning for internet of things: A comprehensive survey, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1622–1658.

[5] E. Borgia, The internet of things vision: Key features, applications and open issues, Comput. Commun. 54 (2014) 1–31.

[6] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: A survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2347–2376.

[7] B.G. Batista, C.H.G. Ferreira, D.C.M. Segura, D.M. Leite Filho, M.L.M. Peixoto, A QoS-driven approach for cloud computing addressing attributes of performance and security, Future Gener. Comput. Syst. 68 (2017) 260–274.

[8] D.M. Leite, M.L.M. Peixoto, C.H.G. Ferreira, B.G. Batista, D.C.M. Segura, M.J. Santana, R.H.C. Santana, A cloud computing price model based on virtual machine performance degradation, Int. J. Comput. Sci. Eng. 18 (4) (2019) 451–463.

[9] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J.P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, J. Syst. Archit. 98 (December 2018) (2019) 289–330.

[10] L.M. Vaquero, L. Rodero-Merino, J. Caceres, M. Lindner, A break in the clouds, ACM SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 50.

[11] G. Gangadharan, Open source solutions for cloud computing, Computer 50 (1) (2017) 66–70.

[12] D.M. Leite, M.L.M. Peixoto, B.G. Batista, B.T. Kuehne, C.H.G. Ferreira, The influence of resource allocation on cloud computing performance, in: Proceedings of the Symposium on Applied Computing, SAC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1516–1521.

[13] B.G. Batista, N.B. Morais, B.T. Kuehne, R.M. Frinhani, D.M. Filho, M.L. Peixoto, Heuristic performance evaluation for load balancing in cloud, in: 2018 International Conference on High Performance Computing & Simulation (HPCS), 2018, pp. 593–600.

[14] M. Peixoto, E. Mota, A. Maia, W. Lobato, M. Salahuddin, R. Boutaba, L. Villas, FogJam: A fog service for detecting traffic congestion in a continuous data stream VANET, Ad Hoc Netw. 140 (2023) 103046.

[15] J. Dogani, R. Namvar, F. Khunjush, Auto-scaling techniques in container-based cloud and edge/fog computing: Taxonomy and survey, Comput. Commun. 209 (2023) 120–150.

[16] C.C.A. Vieira, L.F. Bittencourt, T.A.L. Genez, M.L.M. Peixoto, E.R.M. Madeira, RAaaS: Resource Allocation as a Service in multiple cloud providers, J. Netw. Comput. Appl. 221 (2024) 103790.

[17] A.K. Jumani, J. Shi, A.A. Laghari, Z. Hu, A.u. Nabi, H. Qian, Fog computing security: A review, Secur. Priv. e313,

[18] M. Peixoto, A. Maia, E. Mota, E. Rangel, D. Costa, D. Turgut, L. Villas, A traffic data clustering framework based on fog computing for VANETs, Veh. Commun. 31 (2021) 100370.

[19] M.M. Peixoto, T.A.L. Genez, L.F. Bittencourt, Hierarchical scheduling mechanisms in multi-level fog computing, IEEE Trans. Serv. Comput. 15 (5) (2022) 2824–2837.

[20] F. Faticanti, M. Savi, F. De Pellegrini, D. Siracusa, Locality-aware deployment of application microservices for multi-domain fog computing, Comput. Commun. 203 (2023) 180–191.

[21] H.K. Apat, R. Nayak, B. Sahoo, A comprehensive review on Internet of Things application placement in Fog computing environment, Internet Things (2023) 100866.

[22] D. Charântola, A.C. Mestre, R. Zane, L.F. Bittencourt, Component-based scheduling for fog computing, in: UCC 2019 Companion - Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing, 2019, pp. 3–8.

[23] W. Javed, G. Parveen, F. Aabid, S.U. e Rubab, S. Ikram, K.U.U. Rehman, M. Danish, A review on fog computing for the internet of things, in: 2021 International Conference on Innovative Computing (ICIC), IEEE, 2021, pp. 1–7.

[24] P. Hu, S. Dhelim, H. Ning, T. Qiu, Survey on fog computing: architecture, key technologies, applications and open issues, J. Netw. Comput. Appl. 98 (2017) 27–42.

[25] P. Varshney, Y. Simmhan, Demystifying fog computing: Characterizing architectures, applications and abstractions, in: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), IEEE, 2017, pp. 115–124.

[26] L.F. Bittencourt, J. Diaz-Montes, R. Buyya, O.F. Rana, M. Parashar, Mobility-aware application scheduling in fog computing, IEEE Cloud Comput. 4 (2) (2017) 26–35.

[27] J.K. Zao, T.T. Gan, C.K. You, S.J.R. Méndez, C.E. Chung, Y. Te Wang, T. Mullen, T.P. Jung, Augmented brain computer interaction based on fog computing and linked data, in: 2014 International Conference on Intelligent Environments, IEEE, 2014, pp. 374–377.

[28] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, R. Buyya, iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments, Softw. - Pract. Exp. 47 (9) (2017) 1275–1296.

[29] R. Xu, S.Y. Nikouei, Y. Chen, A. Polunchenko, S. Song, C. Deng, T.R. Faughnan, Real-time human objects tracking for smart surveillance at the edge, in: 2018 IEEE International Conference on Communications (ICC), IEEE, 2018, pp. 1–6.

[30] U. Gawande, K. Hajari, Y. Golhar, Pedestrian detection and tracking in video surveillance system: issues, comprehensive review, and challenges, in: Recent Trends in Computational Intelligence, IntechOpen, 2020, pp. 1–24.

[31] M. Ghobaei-Arani, A. Souri, A.A. Rahmanian, Resource management approaches in fog computing: a comprehensive review, J. Grid Comput. 18 (1) (2020) 1–42.

[32] K. Toczé, S. Nadjm-Tehrani, A taxonomy for management and optimization of multiple resources in edge computing, Wirel. Commun. Mob. Comput. 2018 (2018).

[33] V. Chamola, C.-K. Tham, G.S.S. Chalapathi, Latency aware mobile task assignment and load balancing for edge cloudlets, in: 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), 2017, pp. 587–592.

[34] H. Shah-Mansouri, V.W. Wong, Hierarchical fog-cloud computing for IoT systems: A computation offloading game, IEEE Internet Things J. 5 (4) (2018) 3246–3257.

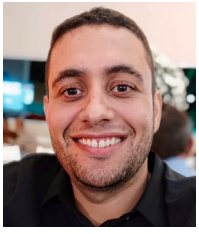
[35] S. Yangui, P. Ravindran, O. Bibani, R.H. Glitho, N. Ben Hadj-Alouane, M.J. Morrow, P.A. Polakos, A platform as-a-service for hybrid cloud/fog environments, in: 2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), 2016, pp. 1–7.

[36] M. Taneja, A. Davy, Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm, in: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2017, pp. 1222–1228.

[37] R.O. Aburukba, M. AliKarrar, T. Landolsi, K. El-Fakih, Scheduling Internet of Things requests to minimize latency in hybrid Fog–Cloud computing, Future Gener. Comput. Syst. 111 (2020) 539–551.

[38] I.M. Ali, K.M. Sallam, N. Moustafa, R. Chakraborty, M.J. Ryan, K.-K.R. Choo, An automated task scheduling model using non-dominated sorting genetic algorithm II for fog-cloud systems, IEEE Trans. Cloud Comput. (2020) 1.

[39] A. Kaur, N. Auluck, O. Rana, Real-time scheduling on hierarchical heterogeneous fog networks, IEEE Trans. Serv. Comput. (2022).



MSc. Leonan Teixeira de Oliveira: received his Maters´ degree in computer science from Federal University of Bahia (UFBA), Brazil, in 2022. He is an Assistance Professor at the Federal Institute of the North of Minas Gerais (IFNMG), Brazil. His research interests include cloud computing, services scheduling and network management.

Dr. Luiz Fernando Bittencourt: is an Associate Professor at the University of Campinas (UNICAMP), Brazil. Luiz was awarded with the IEEE Communications Society Latin America Young Professional Award in 2013. He acts on the organization of several conferences in the cloud computing and edge computing subjects, and in several technical program committees. He also serves as associate editor for the IEEE Cloud Computing Magazine, for the Computers and Electrical Engineering journal, and for the Internet of Things Journal. His main interests are in the areas of resource management and scheduling in cloud, edge, and fog computing.

Dr. Thiago A. L. Genez: is a Software Engineer at the European Bioinformatics Institute (EBI) of the European Molecular Biology Laboratory (EMBL), Hinxton, UK. Prior to this, Thiago was a Research Associate at the University of Cambridge, UK, a Postdoctoral Research Fellow at the University of Bern, Switzerland, and a Research Assistant at Loughborough University, UK. Thiago holds a Ph.D. in Computer Science from the University of Campinas (UNICAMP), Sao Paulo, Brazil. His main research interests are on distributed systems and computer networks, more specifically running workflow-like applications in cloud-based environments.

Dr. Eyal de Lara: is a Full Professor in the Department of Computer Science at the University of Toronto. His focus is on experimental research on mobile and pervasive computing systems. Prof. de Lara served as editor in chief of ACM GetMobile the flagship publication of ACM SigMobile. His research has been recognized with an IBM Faculty Award, an NSERC Discovery Accelerator Award, the 2012 CACS/AIC Outstanding Young Computer Science Researcher Prize, and two best paper awards.

Dr. Maycon Leone Maciel Peixoto: is an Associate Professor at Institute of Computing of Federal University of Bahia (UFBA). He holds a Ph.D. in Computer Science from the University of Sao Paulo (USP), Brazil, 2012 and his Master Degree in Computer Science from the University of Sao Paulo, 2008. He conducted a Post-Doctorate (2020) at the University of Campinas (UNICAMP) in the area of Urban Computing. He serves on the main network and distributed system committees. Over the years, he has participated and coordinated several projects with funding from public funding agencies and also through the private sector.