Future Generation Computer Systems 157 (2024) 51–66

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Towards providing a priority-based vital sign offloading in healthcare with serverless computing and a fog-cloud architecture

Gustavo André Setti Cassel [a], Rodrigo da Rosa Righi [a,*], Cristiano André da Costa [a], Marta Rosecler Bez [b], Marcelo Pasin [c]

[a] *Software Innovation Laboratory (SOFTWARELAB), Applied Computing Graduate Program, Universidade do Vale do Rio dos Sinos, São Leopoldo, RS, Brazil*
[b] *Creative Industry, Feevale, Novo Hamburgo, RS, Brazil*
[c] *Institut d'informatique, University of Neuchâtel, Neuchâtel, Switzerland*

ARTICLE INFO

Keywords:
Healthcare
Internet of Things
Priority
Fog
Cloud
Serverless computing

ABSTRACT

Smart cities and healthcare services have been gaining much attention in recent years, as the benefits offered by this field of research are significant and improve quality of life. Systems can proactively detect health problems by monitoring a person's vital signs and make automated decisions to prevent these problems from worsening. With this in mind, we highlight two essential requirements that smart city architectures should consider to achieve elevated quality of experience in the healthcare domain. The first is to process high-priority vital signs with short response times, so people with comorbidities can have health problems identified as soon as possible. The second is to employ scalability techniques to deal with high usage peaks resulting from people concentrating in specific city neighborhoods. This paper contributes to this field of research by proposing SmartVSO - a computational model of a hierarchic, scalable, tree-based, fog-cloud architecture that executes healthcare services with reduced response time for urgent vital signs. Fog computing is employed to reduce response times, and cloud computing is used to provide virtually infinite computing resources. Serverless computing is the main technology we consider for deploying and running healthcare services because this allows authorized companies to implement their own services in a distributed and pluggable approach, without recompiling the proposed modules and reducing scalability concerns for engineers. An experiment with 80,000 vital signs indicates that our solution processes 60% of the very critical ones in no more than 5.3 s, while an architecture without fog computing and without prioritization takes up to 231 min (around 3 h and 51 min) to process 60% of very critical vital signs.

1. Introduction

The Internet of Things (IoT) has gained much attention in the healthcare domain over the last few years due to the numerous benefits it provides [1,2]. In this context, wearable devices like smartwatches can continuously collect vital signs from the human body, including blood pressure, temperature, and heartbeat rate. This data can be automatically sent to specialized software, known as healthcare service, which analyzes the information and triggers alarms whenever the data undergoes abnormal changes. Among different healthcare services that can be implemented, an example is to automatically call an ambulance when the healthcare service predicts that the person is close to having a heart failure, according to historical data combined with vital signs collected in the last few minutes. Another example is to send notifications to close relatives reporting fever or hypothermia. The union of the healthcare domain with IoT in smart cities may considerably improve the quality of life of the population, and in extreme cases, it can even be a determinant factor in saving people's lives. Healthcare services can predict the occurrence of severe problems before they happen, providing a real-time monitoring of the health status [3–6].

Moreover, cloud computing and fog computing play a fundamental role when it comes to processing ubiquitous IoT data generated by wearable devices. The cloud provides virtually infinite computing resources, important when consuming amounts of IoT data that fluctuate throughout the day. In turn, the fog is an intermediate layer between end-user devices and the cloud. This layer offers computing capabilities physically close to users, resulting in shorter response times when compared to sending data to the cloud or executing tasks on the cloud. Finally, fog and cloud computing complement each other and work

* Corresponding author.
E-mail addresses: gustavoasc@edu.unisinos.br (G. André Setti Cassel), rrrighi@unisinos.br (R. da Rosa Righi), cac@unisinos.br (C. André da Costa), martabez@feevale.br (M. Rosecler Bez), marcelo.pasin@unine.ch (M. Pasin).

https://doi.org/10.1016/j.future.2024.03.032
Received 10 July 2023; Received in revised form 30 January 2024; Accepted 16 March 2024
Available online 21 March 2024

together, providing solutions for smart cities that consider both short response times and scalability techniques.

This context imposes computational challenges that must be satisfied to achieve the desired quality of experience for end-users. On one hand, critical healthcare services may impose latency requirements and should be executed promptly, potentially in the fog layer due to the short physical proximity to the person. On the other hand, machines executing these services have limited computing resources in the fog layer, so only a subset of vital signs can be processed in the fog at the same time. Since people move between neighborhoods of a smart city and may concentrate on specific locations during the day, the fog infrastructure should be able to cope with fluctuations and increases in load. Managing computing resources appropriately, by understanding which vital signs are critical and should be prioritized in the fog, is essential to enable high-quality healthcare services [7]. Scalability, fault tolerance, and mitigation of network congestion are also essential when building healthcare systems, which can be decisive in saving a person's life [8,9].

Conversely, serverless computing is also known as Function as a Service (FaaS) and comes as a promising technology to implement healthcare services. By using serverless computing, business applications can be arranged as a composition of stateless functions, deployed and executed within a sandboxed and isolated environment [10]. These functions are triggered without software engineers worrying about servers. Serverless platforms take the responsibility of automatically spawning function replicas to deal with an increase in incoming requests. In practice, servers do need to exist to execute functions, but from the software engineer's perspective, managing servers is not required. Although originally proposed for the cloud, this technology can also be used in the fog layer. This allows functions to be executed near the users, thus benefitting from the essential reduced latency for the field of healthcare.

To the best of our knowledge, no work in the literature employs a fog-cloud architecture in the healthcare domain as we do, with an offloading strategy inspired by priority levels in emergency triage systems. Related works already propose offloading algorithms in such fog-cloud environments. However, none considers the priority of the healthcare service and the priority of the user simultaneously to offload vital signs among nodes in scalable a fog tree. With this in mind, we propose SmartVSO (Smart Vital Sign Offloading), a model of a scalable fog-cloud architecture for processing vital signs with healthcare services. Our model manages computing resources in the fog satisfying the expected quality of experience for individuals wearing smartwatches. Besides a novel offloading strategy, we consider scalability aspects to allow easy inclusion of fog nodes, while employing serverless computing as a plug-and-play approach to enrich smart cities with new healthcare services. Our main contributions are as follows:

- We introduce a novel recursive vital sign offloading strategy inspired by leading emergency triage systems, combining five user and service priority levels into a ranking that favors vital signs from individuals with health problems monitored by critical healthcare services.
- The proposed fog-cloud architecture combines our novel offloading strategy with serverless computing and existing scalability techniques from the literature, enhancing the quality of experience in smart cities and being determinant in saving people's lives.

This paper is structured as follows. Section 2 explores computational offloading in healthcare, highlighting existing gaps. Section 3 introduces SmartVSO, its algorithms, and the proposed offloading strategy prioritizing vital signs for individuals in critical conditions. Section 4 outlines the evaluation methodology, while Section 5 presents experiment outcomes. Section 6 deliberates on contributions and limitations. Finally, Section 7 summarizes the model's benefits, limitations, and suggests opportunities for future work.

2. Related work

This section explores related work regarding computational offloading between IoT data and fog/cloud layers. Table 1 summarizes selected papers, detailing year, goals, and features. It highlights whether each paper addresses: *(i)* healthcare solutions or prototypes, *(ii)* prioritization strategies for offloading IoT data, and if such strategies consider a priority coming from the user and a priority specific to the service or task consuming the data, *(iii)* serverless computing, *(iv)* edge/fog computing, *(v)* cloud computing, and *(vi)* a tree-based fog node organization. Some papers prioritize vital signs in the healthcare domain, while others employ serverless computing without a healthcare-centric focus, but not both simultaneously. To our knowledge, no paper combines a tree-based fog node structure, an offloading strategy to offload vital signs based on user and healthcare service priorities, and serverless computing for deployment and execution of healthcare services as functions.

In non-healthcare-focused works, Cheng et al. [11] present the Fog Function model for running IoT services on fog and cloud. Bermbach et al. [13] propose an auction-based scheduling algorithm for optimal placement of serverless functions. George et al. [14] introduce IoTPy, a Python runtime system for running serverless functions on edge and IoT devices. Hassan, Azizi, and Shojafar [15] focus on enhancing Quality of Service (QoS) and minimizing energy consumption. Pelle et al. [16] use 5G network telemetry and edge node status to select between edge or cloud for running functions. Dehury et al. [17] propose a Deep Reinforcement Learning (DLR) approach to optimize serverless function placement, minimizing latency while maintaining QoS. Rausch, Rashed, and Dustdar [18] present Skippy, a Kubernetes scheduler for optimizing serverless function placement on edge devices. Cicconetti, Conti, and Passarella [19] offer a decentralized approach for horizontal offloading of serverless functions at the edge. Arora and Singh [21] focus on distributing processing on heterogeneous fog nodes and organizing services by priority. Bukhari et al. [22] propose a task-offloading algorithm for optimal QoS and resource utilization, employing horizontal offloading among fog nodes and vertical offloading between fog and cloud.

In healthcare-focused research, Rezazadeh, Rezaei, and Nickray [12] propose LAMP, a latency-aware algorithm for selecting optimal fog nodes based on resources and delay. AlZailaa et al. [20] develop an algorithm for task scheduling in e-health, prioritizing tasks based on payload semantics. Daraghmi et al. [23] introduce a three-layer architecture for remote health monitoring using NarrowBand IoT (NB-IoT). Hajvali et al. [24] present an IoT-based healthcare architecture with fog and cloud computing, addressing user mobility and non-functional system requirements. Gupta and Chaurasiya [25] propose an IoT-Fog-based health monitoring system with a Bayesian Belief Network. Jasim and Al-Raweshidy [26] present HMAN, a cooperative hierarchical healthcare architecture combining offloading strategies for processing and offloading vital signs on edge, fog, and cloud, but without user and service prioritizations.

3. SmartVSO model

This section introduces SmartVSO — Smart Vital Sign Offloading — a scalable fog-cloud computational model for processing vital signs with healthcare services considering user and service priorities. Design decisions, architecture, scalability techniques, and the offloading strategy will be detailed.

3.1. Design decisions

SmartVSO operates in smart cities, where individuals utilize wearable devices, like smartwatches, to collect vital signs at pre-defined intervals. The mobility of individuals, influenced by their daily work routines and social engagements, is a significant factor we consider

Table 1
Related works on the field of data offloading and service placement for IoT. A subset of works are focused on the healthcare domain, but none of them employed serverless computing to process data. Other works employ offloading techniques with serverless computing but are not focused on the healthcare domain.

Paper	Year	Main goal	Health-	IoT data prioritization		Serverless	Edge/	Cloud?	Tree-based
			care?	User prio.?	Service prio.?	computing?	Fog?		fog nodes?
[11]	2019	Reduce latency and response time	–	–	✓	✓	✓	✓	–
[12]	2019	Select fog nodes based on resources and latency	✓	–	–	–	✓	✓	✓
[13]	2020	Auction-based scheme to process functions on the fog	–	–	–	✓	✓	✓	–
[14]	2020	Reduce response time and reduce energy consumption	–	–	–	✓	✓	✓	–
[15]	2020	Maximize QoS and minimize power consumption	–	–	✓	–	✓	✓	–
[16]	2021	Optimize the deployment of latency-sensitive services	–	–	–	✓	✓	✓	–
[17]	2021	Find optimal deployment of serverless functions	–	✓	✓	✓	✓	✓	–
[18]	2021	Use resources at the edge infrastructure efficiently	–	–	–	✓	✓	–	–
[19]	2021	Distribute serverless functions in a decentralized way	–	–	–	✓	✓	–	–
[20]	2021	Reduce latency for critical tasks	✓	–	✓	–	✓	✓	–
[21]	2021	Distribute heterogeneous services on heterogeneous fog nodes	–	–	–	–	✓	✓	✓
[22]	2022	Maximize QoS and resource usage with logistic regression	–	–	✓	–	✓	✓	–
[23]	2022	Remote health monitoring with NB-IoT communication	✓	✓	–	–	✓	✓	–
[24]	2022	Architecture for IoT-based healthcare	✓	✓	–	–	✓	✓	✓
[25]	2022	Reduce response time with BNN and fog computing	✓	–	–	–	✓	✓	–
[26]	2023	Process vital signs in a cooperative hierarchical architecture	✓	–	–	–	✓	✓	✓
Ours	2024	Process vital signs with priorities in tree-based architecture	✓	✓	✓	✓	✓	✓	✓

when proposing this model that involves neighborhood transitions throughout the day. Healthcare services process collected vital signs, providing timely notifications upon prediction and anomaly detection. In SmartVSO, these services are implemented as serverless functions deployed on fog and cloud, receiving vital signs as input. Fog nodes strategically placed in city neighborhoods minimize communication latency by processing vital signs locally and avoiding unnecessary transmission to the cloud. SmartVSO introduces offloading strategies for dynamic workloads, making placement decisions based on local computing resources and user/healthcare service priorities. This paper assumes the following:

1. Fog nodes have limited computing capabilities, so they can become overloaded as people concentrate on specific neighborhoods and move through the city.
2. Cloud provides virtually infinite computing resources but incurs extra latency due to its physical distance, and only processes vital signs when fog nodes are unable to handle the workload originating from specific neighborhoods.
3. Vital signs from people with health problems should be processed before those from healthy people to enable timely notifications and enhance quality of experience.
4. Discovering which fog node is physically closer to the person is beyond this work's scope, so we assume that fog nodes receive vital signs from people walking nearby.

We also consider Quality of Experience as a human-centric quality aspect, considering user requirements and perceptions, instead of only focusing on a system's perspective guided by technical and telecommunications metrics [27,28]. This is important because users with comorbidities require closer attention than healthy people, thus requiring faster responses.

3.2. Architecture and proposed modules

SmartVSO considers a hierarchical fog computing architecture for arranging fog nodes in the shape of a tree, where fog nodes are distributed among smart city neighborhoods. Fig. 1 illustrates this architecture, clarifying that smartwatches transmit vital signs to the nearest fog node in the initial level of the tree. This is done to minimize communication latency. Meanwhile, the tree is organized in a way that fog nodes are only connected to a single parent. Given this reasoning, a fog node can have multiple children, except for leaf nodes, which never have a child. The root fog node in the hierarchy is the only one connected to the cloud, thus providing virtually infinite computing capabilities when the fog lacks resources to handle large workloads or

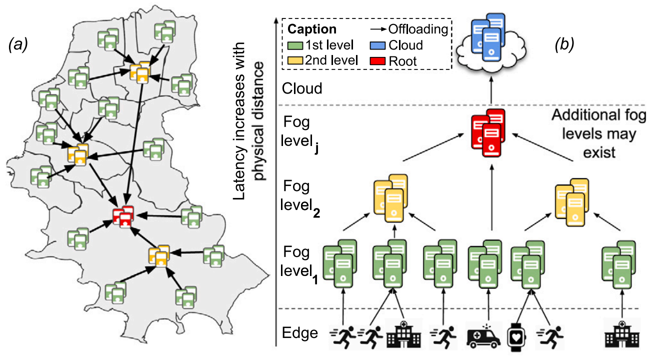


Fig. 1. Fog nodes arranged in the form of a tree. In *(a)*, fog nodes are physically distributed among city neighborhoods. In *(b)*, smartwatches and medical equipment send vital signs to nodes on the first level of the tree, while latency increases when offloading vital signs to higher levels of the architecture.

deal with sudden usage peaks, such as people concentrating in specific neighborhoods.

The main reasons why we decided to use a tree-based fog computing architecture are: *(i)* reduced communication latency due to physical proximity with smartwatches, *(ii)* low computational overhead when making offloading decisions since there is only one parent, *(iii)* ease of including fog nodes on city neighborhoods, and *(iv)* reduced network congestion. The offloading decision is agile because there is only one path to choose when offloading the vital sign, which is the parent fog node, instead of performing extra computation to select which node to offload data. A drawback is the lack of alternative paths to route vital signs when a node faces hardware or communication problems and becomes unreachable, but fault-tolerant techniques could be employed for choosing new parents in such cases.

Moreover, scalability is important for SmartVSO and can be defined as the ability of a system to cope with an increase in load [29]. In terms of the healthcare domain, this translates to a rise in the number of individuals utilizing healthcare systems and wearing smart devices, such as smartwatches, resulting in additional vital signs being processed. SmartVSO achieves scalability in two steps. The first is via communication between child and parent fog nodes in the fog tree. The second is via fog-cloud communication, as the fog node located in the root of the tree is connected to the cloud, which provides virtually infinite computing capabilities when the fog is overloaded.

As more people decide to use smartwatches and have their vital signs monitored by healthcare services, the fog infrastructure needs to

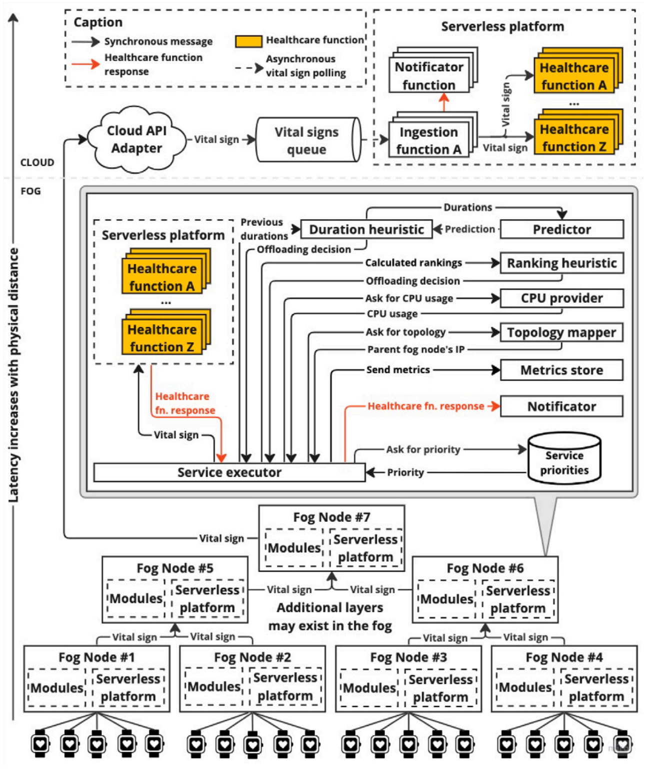


Fig. 2. Interaction between hierarchical fog nodes and the cloud, while each node runs a replica of the proposed modules. Each node is directly connected to its parent, and the last node on the hierarchy is connected to the cloud.

expand to cope with such an increase in load, while still guaranteeing short response times. SmartVSO supports this scenario by allowing extra fog nodes to be easily included in the architecture at any time, thus increasing the computing capacity near the users. We emphasize that the cloud is already able to handle dynamic workloads, but has the disadvantage of experiencing increased latency due to the distant physical location. In case the fog is constantly overloaded, it should be empowered with more fog nodes to reduce such additional latency caused by offloading vital signs to the cloud.

In turn, Fig. 2 presents the main modules introduced in this architecture. Each fog node runs a replica of every module, meaning that in a smart city composed of x fog nodes there will be x replicas of every module running on each node. The *service executor* is the entry point for ingesting vital signs. It is responsible for interacting with most of the other modules and deciding between processing the vital sign locally, with a healthcare service implemented as a serverless function, or offloading the vital sign to the *service executor* module running in the parent fog node. This module interacts with *CPU provider*, which asynchronously collects the CPU usage on the local fog node and provides the last observation when this module is invoked. This information is used in the offloading decision by the *service executor*. Furthermore, the *ranking heuristic* and *duration heuristic* modules make offloading decisions when the CPU utilization in the fog node is between predefined lower and upper thresholds. The *predictor* module is used by the *duration heuristic* to forecast how long the healthcare service will take to complete in its next execution. Additionally, each fog node runs a *metrics store*, responsible for storing and providing metrics during the execution of healthcare services and the ingestion of vital signs. This module is used in this paper to generate charts after running experiments, but future work can use it to monitor in real-time how fog nodes are behaving and how many vital signs are received in each neighborhood.

Still in Fig. 2, the *topology mapper* is responsible for resolving the mapping between the current fog node and the upper layer in the tree, which can be the parent fog node or the cloud. An existing challenge is how the current fog node determines the destination to which it should offload vital signs, but *topology mapper* solves this with a global view of the connection between fog nodes. The *notificator* module is located in both fog and cloud layers to send notifications to the user, depending on the result of healthcare functions processed either locally or in the cloud. This module is out of the scope of this paper and is presented for contextualization purposes. Finally, the module named *Cloud API Adapter* runs in the cloud and is the entry point for ingesting vital signs in this layer. It receives vital signs offloaded from the last fog node in the hierarchy and stores them in a persistent queue. These vital signs are processed asynchronously when serverless function replicas become available in the cloud serverless platform. On purpose, this module implements the same contract (API) as the *service executor* that runs on each fog node, ensuring a generic architecture. Once the fog node acquires its parent's IP address, it does not need to discern if the parent is another fog node or the cloud. They have the same API, so the root fog node offloads the vital sign to its parent (the *Cloud API Adapter*) and benefits from virtually infinite computing resources, without even knowing it was offloaded to the cloud.

Moreover, Fig. 3 displays a flow chart that outlines the key steps involved in the offloading strategy that will be further detailed in Section 3.6.3. This process happens recursively until a fog node with sufficient processing capacity is identified. While this diagram focuses on the fog layer for the sake of simplicity, vital signs can also be offloaded to the cloud by the root fog node in the tree. This imposes a longer response time due to the increased latency, but benefits from virtually infinite processing capabilities. In other words, if the vital sign cannot be processed on the local fog node, it is recursively offloaded to a *service executor* replica in a parent fog node, or to the *Cloud API Adapter* if the upper layer is the cloud.

3.3. Smartwatch integration

Collecting vital signs from smartwatches and sharing them with fog nodes is essential for SmartVSO. Different architectures can be employed for this communication, but sending vital signs to the *service executor* module located in the fog node is a premise that must be satisfied. Moreover, smartwatches from particular vendors collect vital signs and transmit them in specific formats, via specific protocols and communication channels. Interoperating among formats is challenging but still required to enhance the adherence of SmartVSO in smart cities, where people can wear smartwatches from different vendors.

Potential approaches to communicating vital signs include edge devices translating heterogeneous IoT protocols and message formats into a unique format understood by SmartVSO. Smartwatches could collect vital signs and share them with the user's smartphone via Bluetooth or similar technology that does not involve sending data to the cloud. This way, smartphones can be in charge of transmitting these vital signs to the closest fog node. Another approach is having edge devices located at home, which play the role of smart gateways receiving vital signs directly from smartwatches and retransmitting them to the closest fog node, without relying on the smartphone. Finally, extra features can be implemented in gateways and intermediate edge devices to enrich the smartwatch integration. Examples include aggregating and compacting data to reduce the size of the messages transmitted via the network, helping save bandwidth and battery. Moreover, encryption is essential to prevent vital signs from being stolen or modified in transit. This should occur prior to vital signs reaching the initial fog node in the tree, as well as during offloading operations, where vital signs are transferred from the current fog node to its parent or the cloud. We designate this as future work, with our primary emphasis on the architecture and offloading strategy.

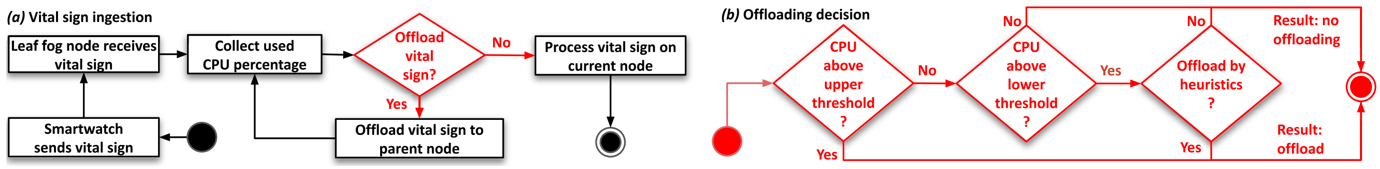

(a) Vital sign ingestion
Leaf fog node receives vital sign
Collect used CPU percentage
Offload vital sign?
No
Process vital sign on current node
Yes
Smartwatch sends vital sign
Offload vital sign to parent node
(b) Offloading decision
CPU above upper threshold ?
No
CPU above lower threshold ?
Yes
No
Offload by heuristics ?
No
Result: no offloading
Yes
Yes
Result: offload

Fig. 3. Flow diagrams presenting how vital signs are ingested by SmartVSO. In *(a)*, the vital sign is ingested by the leaf fog node and is recursively offloaded as needed until it can be processed locally. In *(b)*, internals of the offloading decision are presented. The offloading strategy will be deeper detailed in Section 3.6.3.

3.4. Anatomy of a healthcare service

Healthcare services are developed as serverless functions that receive vital signs as their input. In this paper, the term *healthcare service* is interchangeably used as a synonym for *serverless function*. The selected technology offers the advantage of enabling authorized software companies to develop and introduce new healthcare services for the smart city. This can be achieved without recompiling SmartVSO, thus streamlining the process and promoting greater ease of use. Additionally, software houses can focus as much as possible on the healthcare domain and do not need to focus on scalability concerns and placement decisions between fog nodes and the cloud, since this is SmartVSO's responsibility.

Fig. 4(a) presents a healthcare service implemented as a serverless function with the Python programming language, although other languages can also be used. The entry point for the execution is *handle*, while the *req* parameter contains the vital sign to be ingested. This specific healthcare service deserializes the vital sign from a text-based *JSON* message and checks if the person has fever or hypothermia, by comparing the temperature against thresholds. The response only indicates that the user should not be notified in case the temperature is between acceptable margins. Otherwise, if the person has fever or hypothermia, a different response alerts the person with a message containing the detected problem. Fig. 4(b) presents a sample message as the input of healthcare services, which is a vital sign collected from an individual using a smartwatch.

We emphasize that the healthcare service presented in Fig. 4(a) only analyzes the temperature, but others services could detect if a person is close to having a heart failure by analyzing multiple data, for example. Pre-trained Artificial Intelligence (AI) models can also be invoked from healthcare services, which provides great flexibility in how the service can be implemented and what features can be provided to users in a smart city. Finally, individuals may utilize smartwatches from various vendors, and each smartwatch may collect specific vital signs from the human body. Therefore, production-ready services should not assume that a given data is always available in the received message because this is not guaranteed.

3.5. Deployment strategy

As mentioned in Section 3.4, healthcare services are implemented as serverless functions. This allows great extensibility to the architecture, by easily enriching the smart city with new functions that monitor vital signs for different purposes. Therefore, whenever a healthcare service is created, or an existing service is changed, it must be deployed to the serverless platform running in the cloud, which consumes vital signs that were not processed by fog nodes, and also to the serverless platform running in each fog node. Let f be the number of fog nodes distributed in the smart city, and c the number of serverless platforms in the cloud, which in the version proposed by this paper is always $c = 1$. The healthcare service then needs to be deployed to $f + c$ different serverless platforms. We highlight that in SmartVSO the fog nodes work as independent units and only communicate with each other when it needs to offload vital signs. There is not a single serverless platform shared among fog nodes, but multiple serverless platform replicas instead, each running in a different fog node.

(a)

```
import json

def alert(msg):
    return json.dumps({"notify": True, "msg": msg})

def handle(req):
    vital_sign = json.loads(req)
    if vital_sign["temperature"] > 38:
        return alert("Has␣fever")
    elif vital_sign["temperature"] < 35:
        return alert("Has␣hypothermia")
    else:
        return json.dumps({"notify": False})
```

(b) `{"heartbeat": 97, "spo2": 99, "temperature": 40}`

Fig. 4. In *(a)*, the source code of a healthcare service implemented as a serverless function that receives a vital sign, processes it, and returns a JSON telling whether the person has fever or hypothermia. In *(b)*, an example of a vital sign collected from a smartwatch, feeding the healthcare function.

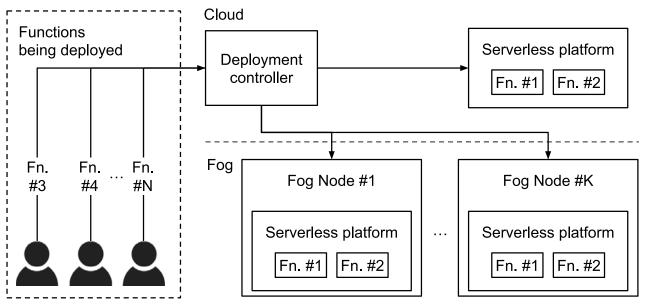


Fig. 5. Deployment strategy for healthcare services. In this example, functions #1 and #2 have already been deployed, while functions #3 through #N are currently being deployed.

Fig. 5 illustrates the deployment strategy for SmartVSO. In the long term, particularly in large smart cities, manually deploying healthcare services to each fog node becomes impractical. A cloud-based deployment controller, equipped with a comprehensive view of the fog infrastructure and administrative access to the cloud's serverless platform, is crucial. This controller serves as the entry point for deploying healthcare services as serverless functions, releasing the latest function version across all fog nodes and the cloud. Artifacts, including binaries and dependencies, should be submitted to this component via web requests. Furthermore, the controller module can accommodate extra features, such as removing an experimental healthcare function.

3.6. Vital signs offloading: blending vital signs and healthcare services to improve quality of experience

In this subsection, we provide a comprehensive explanation of the process through which vital signs are offloaded in SmartVSO. We elaborate on user and service priorities, the employed offloading strategy, and heuristics to decide which subset of vital signs should be offloaded or processed locally.

Table 2
Priorities considered in the SmartVSO model. The combination of user and service priorities coming from different sources represents the *ranking*.

Priority	Source	Description
User	Request	Tells how critical the vital sign is, directly related to the health status of the person.
Service	Database	Tells how critical the healthcare service is, without considering any user information.

3.6.1. User and service priorities

To better comprehend the need for priorities, we first emphasize the social context in which SmartVSO is situated. People of various ages and health statuses use smartwatches, so vital signs are collected and systems send notifications whenever health problems are identified. The number of vital signs received by each fog node is dynamic because it depends on the concentration of people in each neighborhood, which fluctuates during the day as people move. With this in mind, two main properties need to be satisfied to increase the Quality of Experience in the context of healthcare for smart cities:

- Users with comorbidities have a stronger requirement to receive prompt notifications about their health statuses, which should be faster than healthy users.
- Computing resources in the fog are limited, so they must be efficiently used to enable prompt notifications for unhealthy users, even when becoming overloaded.

The main challenge is how to use fog nodes when computing resources are getting overloaded, while still satisfying the needs of people in urgent conditions and dealing with scalability problems when people concentrate in specific neighborhoods. People with comorbidities should receive notifications with the lowest possible response time by having vital signs processed in the closest computing infrastructure. At the same time, it is acceptable that healthy users wait for additional seconds or minutes to receive responses, to the detriment of urgent vital signs. Table 2 presents the types of priorities that SmartVSO considers when determining the importance of the vital sign, with details presented in the following paragraphs.

User Priority: analogous to an individual's health state, user priority determines the urgency of vital sign processing. The unhealthier the person, the quicker vital signs should be handled. Timely notifications are crucial for individuals when their health worsens or anomalies are detected in vital signs. Conversely, health states are categorized into five groups with specific requirements for the speed of vital sign processing. This dictates how SmartVSO should use the fog infrastructure to process vital signs while satisfying requirements demanded by the individual's health state. For instance, healthcare services must promptly notify those with diseases if their conditions worsen, while it is acceptable to delay processing vital signs from healthier individuals. In SmartVSO, the primary role of user priority is to guide vital signs routing among fog nodes. The offloading strategy will be explained in Section 3.6.3, but briefly, vital signs for healthy individuals could be offloaded to the parent fog node or the cloud, and the extra delay would not harm healthy individuals. This helps preserve resources in the current fog node for urgent cases. Conversely, individuals with health issues, having higher user priority, have their vital signs processed locally to avoid delays from offloading operations.

Fig. 6 (a) illustrates a sample message expected by a fog node in SmartVSO. This clarifies that the value for the user priority comes from the request payload in JSON format, along with the collected vital sign and the name of the healthcare service that will process this vital sign. The user priority is dynamic in the sense that represents the person's health state at the moment the vital sign was collected. For instance, if an individual feels well during collection, the user priority in the JSON message is low. If the person becomes sick the next day, the user priority value for the vital signs collected on the next day should

```
(a) {"vital_sign": "{\"heartbeat\": 97, \"spo2\": 99,
\"temperature\": 40}", "user_priority": 3, "id":
"4cd7df5c-6056-42c3-8783-7254a2b734fc", "service_name":
"heart-failure-predictor"}

(b) {"vital_sign": "{\"heartbeat\": 97, \"spo2\": 99,
\"temperature\": 40}", "user_priority": 3, "id":
"4cd7df5c-6056-42c3-8783-7254a2b734fc"}
```

Fig. 6. Sample inputs expected by SmartVSO. These messages contain the vital signs collected from the smartwatch, in addition to the user priority, the message's unique identifier, and optionally the name of the healthcare service.

increase. The more critical the health state, the sooner the vital signs are processed, favoring local fog node processing over offloading. Fig. 6 (b) shows another example but this time omits the healthcare service's name in the JSON message, which will execute all deployed healthcare services passing the vital sign as input.

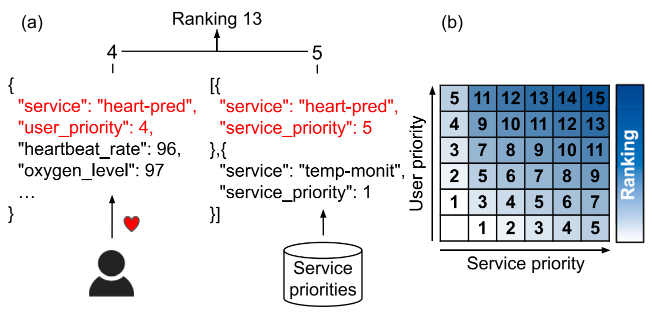
In its current version, SmartVSO, as described in this paper, keeps simplicity by directly accepting the user priority value from received messages without extra computation. Here, the smartwatch is responsible for including user priority in the message. As a future improvement, we suggest integrating a module into SmartVSO to dynamically adjust user priority based on healthcare service results, alleviating smartwatches from this responsibility. For instance, anomalies detected by healthcare services could prompt a higher user priority for closer monitoring. However, this automatic recalibration is beyond the paper's scope. Presently, SmartVSO utilizes the pre-calculated user priority from the request payload as an additional JSON field, along with the vital sign.

Service Priority: the second priority type considered by SmartVSO when routing vital signs among fog nodes. Medical analysis may highlight specific healthcare services as more critical than others in smart city contexts. Given the diverse range of implemented services, it is essential for SmartVSO to prioritize executing critical healthcare services over less critical ones, particularly during peak usage leading to overloaded computing resources. For example, a service predicting imminent heart failure might take precedence over a service monitoring body temperature for fever notifications. When resources are scarce, SmartVSO prioritizes critical services, relegating less critical ones to higher levels in the fog tree. We emphasize that healthcare service priority in SmartVSO should be pre-configured by a group of experienced doctors and fetched from the database, not automatically calculated. At the moment, SmartVSO exposes an HTTP API (Application Programming Interface) to configure these priority values, but a user interface could be built upon this API. Security measures are crucial to prevent unauthorized changes to healthcare service priorities, as such actions could compromise offloading operations and impact public health.

3.6.2. Ranking combining priority values

SmartVSO combines both user priority and healthcare service priority into a single value named *ranking* whenever a vital sign is received by the user's closest fog node. The ranking is the key to deciding if the vital sign should be processed locally, in the current fog node, or offloaded to the parent fog node in the tree, thus reserving computing resources for the most urgent data. The higher the ranking, the higher the importance of the vital sign, and increased the chance to process this vital sign in a fog node close to the first level of the tree and close to the individual. This behavior helps process urgent vital signs with short response time to enhance the quality of experience, due to the physical proximity to the user and the reduced offloading operations between fog nodes.

We consider five levels of priority for the user: (1) *Very healthy*, (2) *Healthy*, (3) *Warning*, (4) *Critical*, and (5) *Very critical*. The reason why we consider five levels is that leading emergency triage systems also consider this number, such as the Canadian Triage and Acuity Scale (CTAS), the Emergency Severity Index (ESI), the Australasian Triage



5	11	12	13	14	15
4	9	10	11	12	13
3	7	8	9	10	11
2	5	6	7	8	9
1	3	4	5	6	7
	1	2	3	4	5

Fig. 7. Rankings calculated from user and healthcare service priorities. In *(a)*, user priority comes from the request payload, and service priority comes from the database. In *(b)*, a matrix with all possible rankings calculated with Eq. (1), considering five priorities and double weight to the user priority.

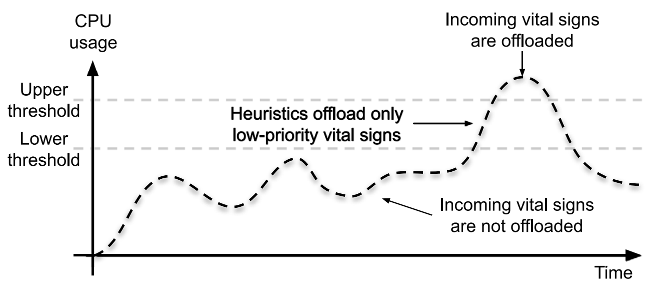


Fig. 8. Strategy to offload vital signs according to the CPU utilization. Static thresholds dictate the offloading decisions, but heuristics are triggered to offload non-critical vital signs when CPU utilization is between these thresholds.

Scale (ATS), and the Manchester Triage System (MTS) [30]. We also consider five priority levels for services: (1) *Low urgency*, (2) *Medium urgency*, (3) *Important*, (4) *Urgent*, and (5) *Very urgent*.

Eq. (1) calculates the ranking taking both priorities into account, while the value of two is attributed to γ to amplify the user priority's influence in the result. We understand that vital signs from a person with health problems should take precedence over those from a healthy person, even when vital signs from this healthy person are processed by an urgent service, and those from the unhealthy person are processed by a non-urgent service. User priority alone is not the sole factor; healthcare service priority becomes decisive when user priorities are the same for different individuals.

$$ranking = \gamma * user\_priority + service\_priority \qquad (1)$$

where:

$$\gamma = 2$$
$$user\_priority \in \{1, 2, 3, 4, 5\}$$
$$service\_priority \in \{1, 2, 3, 4, 5\}$$

To justify assigning double weight to user priority, consider a scenario where a fog node faces overload from simultaneous vital signs. Some of them are from users with severe health issues (level 5), processed by a non-critical service for fever detection (level 1), resulting in a ranking of 11. Others are from very healthy users (level 1), processed by a critical service for heart failure detection (level 5), resulting in a ranking of 7. The double weight on user priority favors emergency cases, increasing their ranking. This boosts the chance of processing their vital sign locally instead of performing an offloading operation, reducing response time. If equal weights were assigned, SmartVSO would be unable to distinguish, yielding the same ranking for both cases and hindering offloading decisions.

Fig. 7*(a)* expands this idea and presents another example of a user in a critical state (priority level 4) having vital signs processed by the very urgent healthcare service *heart-pred* (priority level 5). The priority associated with this healthcare service was previously defined by a group of doctors and therefore comes from the database, resulting in the ranking of 13 with Eq. (1). Fig. 7*(b)* indicates every possible ranking calculated with Eq. (1), taking user and service priorities into account. Values 3 and 15 represent the minimum and maximum rankings calculated with this equation, respectively, while giving double weight to the user priority. Dark blue squares indicate the highest rankings: a vital sign of an individual with serious comorbidities processed by a critical service. Squares in white represent the lowest rankings, calculated for vital signs from very healthy users processed by non-critical services.

3.6.3. Offloading strategy

SmartVSO introduces a strategy to favor vital signs associated with the highest rankings, which are those collected from people with comorbidities and that are processed by critical healthcare services. Conceptually, an offloading operation means forwarding a vital sign from the current fog node to its parent on the tree. This parent can be either another fog node or the cloud, so healthcare functions deployed in a remote machine will further process this vital sign. We emphasize that computing resources on fog nodes are limited, so processing all vital signs locally is not always possible. Deciding which subset of vital signs should be offloaded is crucial. Conversely, offloading operations incur extra network latency and should be avoided whenever possible for critical vital signs — as people in emergency need to be closely monitored and receive prompt notifications. The proposed offloading strategy increases the Quality of Experience by *(i)* processing urgent vital signs locally most of the time, and *(ii)* forwarding non-urgent ones to remote machines, with the following main benefits:

- Urgent vital signs will be processed with reduced response times because non-critical ones will be offloaded to a remote machine, resulting in more computing resources available on the current fog node, although non-critical vital signs will be harmed by the extra network latency.
- Scalability is achieved by allowing more machines to extend the computing capabilities of the architecture, which helps to deal with scenarios where people concentrate in specific neighborhoods and the local machine cannot process such a large number of incoming vital signs.

The extra latency is typically not a problem for non-critical vital signs because healthy people have fewer chances of having severe health problems in the upcoming instants. On the other hand, the extra latency can be a real problem for critical vital signs because each second matters for a person with serious health problems or in emergencies. This can be decisive in saving their lives and is why we favor urgent vital signs by processing them on the local fog node whenever possible. However, there are situations where the current fog node is so overloaded that it cannot process any other incoming vital sign, regardless of being urgent or not. This leads SmartVSO to offload any incoming vital signs when the machine is running out of resources. In the current version, rules for triggering the offloading heuristics depend solely on the percentage of used CPU, although other metrics can be included in the future.

Fig. 8 elucidates how an arbitrary fog node ingests vital signs according to the CPU load, supported by static, pre-defined lower and upper thresholds that guide the offloading strategy. All vital signs can be processed locally when the CPU utilization is below the lower threshold, regardless of the vital sign's priority. The machine still has many computing resources available to handle a large number of vital signs, so processing all of them locally improves the quality of experience even healthy people, who will also receive prompt notifications. No offloading heuristic needs to be triggered in this scenario to favor urgent vital signs or healthcare services that execute fast.

Conversely, only vital signs associated with high rankings are processed locally when the CPU utilization is between the lower and upper thresholds. The goal is to reserve and allocate computing resources to process the subset of the vital signs originated from people with health problems, so they can continue receiving timely notifications. Incoming vital signs from healthy people will be offloaded to the parent node in the hierarchy in favor of urgent vital signs, albeit harmed by extra network latency. The Ranking Heuristic, as presented in Section 3.6.5, is triggered in this context to make the offloading decision by favoring people in urgent conditions. In cases of ranking collisions, the healthcare service duration is also taken into account as a fallback, determining offloading by applying the Duration Heuristic as detailed in Section 3.6.6.

Finally, when the CPU is above the upper threshold, it means that computing resources on the current fog node are overloaded and unable to process incoming vital signs, regardless of their priority. No offloading heuristic is triggered and all incoming vital signs are offloaded to the parent node in the hierarchy — even vital signs from people in critical scenarios will be harmed by extra network latency caused by the offloading operation.

We emphasize that this offloading strategy does not consider a global view of resources in the fog tree, being limited to local information of the current fog node instead. Exchanging metrics between nodes is useful in specific scenarios, such as when there is a sudden surge in demand for healthcare services in a specific neighborhood, but incurs extra overhead. The offloading decision should impose as little overhead as possible, so most processing efforts are dedicated to healthcare services instead of placement decisions.

As a specific example, consider an industrial neighborhood with a single fog node, which is preconfigured with lower and upper CPU thresholds of 60% and 90%, respectively. During night hours, when only a few vital signs are received due to fewer people in this neighborhood, the fog node's CPU remains idle. Therefore, all vital signs can be processed locally, including those from people in healthy conditions. In contrast, during daytime business hours, many people concentrate in this neighborhood, resulting in increased vital sign inputs that elevate CPU usage to a level between the pre-configured lower and upper CPU thresholds. In this scenario, only vital signs from individuals in urgent conditions are processed locally because heuristics are invoked, while those from healthy individuals are offloaded to the parent node in the arranged tree infrastructure. Now, suppose a significant event attracts a large crowd to the neighborhood, causing the CPU usage to surpass the upper threshold. In this case, all incoming vital signs are offloaded to the parent fog node, as the machine exhausts its resources.

3.6.4. Offloading algorithms

Algorithm 1 Entry point of vital sign ingestion in the fog node.

```
Require: $message$
1: procedure $ingest\_vital\_sign\_message(message)$
2:     $user\_priority \leftarrow message.user\_priority$
3:     $vital\_sign \leftarrow message.vital\_sign$
4:     $healthcare\_service \leftarrow message.healthcare\_service$
5:     $ranking \leftarrow calculate\_ranking(user\_priority, healthcare\_service)$
6:     if $should\_offload(ranking, healthcare\_service)$ then
7:         $offload(message)$
8:     else
9:         $id \leftarrow message.id$
10:        $register\_execution\_started(id, healthcare\_service, ranking)$
11:        $response \leftarrow execute\_healthcare\_service(healthcare\_service, vital\_sign)$
12:        $register\_execution\_finished(id)$
13:        $send\_response\_to\_notificator\_module(id, response)$
14:    end if
15: end procedure
```

Exploring the offloading strategy, Algorithm 1 outlines the functioning of the *service executor* module when a vital sign emerges in the fog node. The algorithm, acting as the entry point, receives a message composed of *(i)* the user priority (ranging from one to five), *(ii)* the collected vital sign, and *(iii)* the name of the healthcare service that will process the vital sign. The initial step calculates the vital sign ranking based on user and healthcare service priorities using the equation in Section 3.6.2. Subsequently, Algorithm 2 is invoked with this ranking and the healthcare service name to determine if offloading is required. If required, the input message is recursively forwarded to the parent node until processing can occur in a fog node with available resources or reaches the cloud. Conversely, if not required, the vital sign is locally processed with the designated healthcare service. The current execution duration is recorded, and the service response is relayed to the *notificator* module, facilitating health problem alerts for users.

Algorithm 2 Offloading decision according to: *(i)* percentage of CPU usage, and *(ii)* Ranking and Duration heuristics.

```
Require: $calculated\_ranking, healthcare\_service$
Ensure: $offload\_vital\_sign$
1: function $should\_offload(calculated\_ranking, healthcare\_service)$
2:     $offload\_vital\_sign \leftarrow false$
3:     $cpu\_usage \leftarrow get\_used\_cpu\_percentage()$
4:     if $cpu\_usage > UPPER\_THRESHOLD$ then
5:         $offload\_vital\_sign \leftarrow true$
6:         return $offload\_vital\_sign$
7:     end if
8:     if $cpu\_usage > LOWER\_THRESHOLD$ then
9:         $all\_rankings \leftarrow get\_rankings\_for\_vital\_signs\_being\_processed()$
10:        $decision \leftarrow ranking\_heuristic(calculated\_ranking, all\_rankings)$
11:        if $decision == NON\_DECISIVE$ then
12:            $durations\_service \leftarrow get\_durations\_for(healthcare\_service)$
13:            $durations\_others \leftarrow get\_durations\_ignoring(healthcare\_service)$
14:            $decision \leftarrow duration\_heuristic(durations\_service, durations\_others)$
15:        end if
16:        if $decision == OFFLOAD$ then
17:            $offload\_vital\_sign \leftarrow true$
18:        end if
19:    end if
20:    return $offload\_vital\_sign$
21: end function
```

Algorithm 2 is invoked by the previous algorithm and follows the strategy presented in Section 3.6.3, determining whether the vital sign should be offloaded or processed locally. The algorithm receives as the input *(i)* the ranking previously calculated for the incoming vital sign, and *(ii)* the name of the healthcare service that is supposed to process the vital sign on the current machine. The received ranking feeds the input of the Ranking Heuristic, while the healthcare service name is used to fetch previous execution durations that will feed the input of the Duration Heuristic, in case it needs to be triggered. The algorithm starts by capturing CPU usage and comparing it with predefined thresholds to decide between local processing or offloading. When CPU utilization falls between the thresholds, the Ranking Heuristic is activated to decide the offloading based on rankings of other vital signs being processed in the current fog node. If the heuristic cannot decide, the Duration Heuristic serves as a fallback, considering healthcare service durations and prioritizing healthcare services that complete faster.

The reasons fo triggering the Ranking Heuristic before the Duration Heuristic, and not the opposite, are twofold. The first is that the former is directly related to the person's health status, therefore helping to achieve faster response times for people in urgent conditions. The second is that this heuristic is faster than the Duration Heuristic, as the latter needs an extra step of making predictions, as further shown in Section 3.6.6.

3.6.5. Ranking heuristic

The Ranking Heuristic, in Algorithm 3, makes offloading decisions based on the rankings of vital signs processed on the local fog node. A higher ranking represents a more critical vital sign, potentially from someone with health problems. The algorithm's rationale is to favor vital signs with higher rankings by processing them locally, thus minimizing offloading operations that would result in extra network latency. In other words, offloading is avoided for vital signs from individuals with health issues, ensuring fast processing. The algorithm receives *(i)* the ranking calculated for the received vital sign, and *(ii)* a list with rankings for vital signs currently processed in the local fog node. It starts by sorting and deduplicating the rankings, followed by identifying the median value from this fluctuating list. This adaptability reflects dynamic scenarios where individuals transition between city neighborhoods.

The next step is to compare the ranking calculated for the received vital sign (received as the first argument) with the identified median. If lower than the median, offloading occurs, meaning the received vital sign has lower urgency compared to others. This is analogous to receiving a vital sign from a healthy person in a neighborhood with a hospital, where most individuals have health issues. If higher than the median, local processing ensues, akin to receiving a vital sign from an individual with health problems in a predominantly healthy neighborhood. Finally, if the calculated ranking matches the median, the algorithm defers the offloading decision to the Duration Heuristic. This scenario is comparable to receiving a vital sign from a healthy person in a neighborhood where most individuals are also healthy, with the same user priority.

Algorithm 3 Ranking Heuristic to make offloading decisions based on rankings for vital signs processed on the fog node.

```
Require: $calculated\_ranking, all\_rankings$
Ensure: $decision$
1: function $ranking\_heuristic(calculated\_ranking, all\_rankings)$
2:     $sorted\_rankings \leftarrow sort\_removing\_duplicates(all\_rankings)$
3:     $middle\_ranking \leftarrow median(sorted\_rankings)$
4:     $decision \leftarrow RUN\_LOCALLY$
5:     if $calculated\_ranking < middle\_ranking$ then
6:         $decision \leftarrow OFFLOAD$
7:     end if
8:     if $calculated\_ranking == middle\_ranking$ then
9:         $decision \leftarrow NON\_DECISIVE$
10:    end if
11:    return $decision$
12: end function
```

3.6.6. Duration Heuristic

The Duration Heuristic, in Algorithm 4, guides the offloading decisions based on predictions of how long healthcare services will take to complete, derived from historical service durations. This heuristic solely considers service duration and is independent of user or service priority. The rationale is to favor fast healthcare services by processing them on the current fog node, thus avoiding offloading operations and therefore reducing response time. While extra latency is more noticeable in short-running services, it tends to vanish for longer executions. Running short-term healthcare services at lower levels of the fog tree ensures brief CPU utilization, enabling prompt resource reallocation to process additional vital signs. The algorithm receives *(i)* a list of previous durations for the service that will process the received vital sign, and *(ii)* a matrix where each row corresponds to a list of previous durations for the other healthcare services running on the local fog node.

Algorithm 4 Duration Heuristic to make offloading decisions based on how long healthcare services take to complete, considering predictions based on historical durations.

```
Require: $durations\_target\_service, durations\_running\_services$
Ensure: $decision$
1: function $duration\_heuristic(durations\_target\_service, durations\_running\_services)$
2:     $prediction\_target\_service \leftarrow HES(durations\_target\_service)$
3:     $N \leftarrow length(durations\_running\_services)$
4:     $predictions\_running\_services \leftarrow [N]$
5:     for $i \leftarrow 0$ to $N-1$ do
6:         $durations\_current\_service \leftarrow durations\_running\_services[i]$
7:         $predictions\_running\_services[i] \leftarrow HES(durations\_current\_service)$
8:     end for
9:     $sorted\_predictions \leftarrow sort\_removing\_duplicates(predictions\_running\_services)$
10:    $middle\_prediction \leftarrow median(sorted\_predictions)$
11:    $decision \leftarrow RUN\_LOCALLY$
12:    if $prediction\_target\_service > middle\_prediction$ then
13:        $decision \leftarrow OFFLOAD$
14:    end if
15:    if $prediction\_target\_service == middle\_prediction$ then
16:        $decision \leftarrow NON\_DECISIVE$
17:    end if
18:    return $decision$
19: end function
```

The first step is to flatten all duration values, converting a set of durations into a single value which is the estimated time required to execute the healthcare service. To be precise, the first argument will be transformed from a list of durations to a single numerical value, and the second argument flattens from a matrix into a single list. The *HES* function employed in this algorithm is Holt's Exponential Smoothing because it considers trends and gives exponentially higher weight to more recent observations [31]. Also, HES is computationally efficient as we feed it with only the last six observations to achieve appropriate results, not imposing heavy overhead in the decision-making.

The next steps resemble those in the Ranking Heuristic. The algorithm sorts and deduplicates predictions, then calculates the median. This median serves as a reference for comparing the predicted duration of the healthcare service that will process the received vital sign. If the predicted duration is lower than the median, the service is executed locally for optimized throughput, as it is expected to finish faster than other local services. Conversely, if the predicted duration exceeds the median, offloading to the parent fog node is preferable to accommodate longer processing times. When the predicted duration equals the median, the heuristic cannot determine the offloading operation. This leads SmartVSO to process the vital sign locally as a fallback, thus avoiding extra network latency.

We understand that various factors can affect the processing time of healthcare services, but SmartVSO prioritizes fast decision-making over absolute accuracy. This heuristic deliberately overlooks additional variables to provide prompt responses, albeit potentially less precise. This trade-off between precision and performance is a necessary compromise in meeting the system's objectives of ingesting urgent vital signs in smart cities.

4. Evaluation methodology

This section presents the methodology employed to evaluate SmartVSO. Implementation details, infrastructure for tests, architectures arranged for the experiments, strategy for vital signs generation, and evaluation scenarios are also presented.

4.1. Implementation details

There is no widely used benchmark to evaluate models such as SmartVSO in the literature, combining serverless computing, smart cities, and healthcare. We implemented modules available on vital-signs-ingestion[1] repository on GitHub. The following healthcare services were implemented as serverless functions with Python[2]: *(i) body-temperature-monitor*, to check for fever and hypothermia based on temperature thresholds, and *(ii) heart-failure-predictor*, to predict potential heart failure comparing the oxygen level and the heartbeat to thresholds. We emphasize that healthcare services need medical validation before using in real-world scenarios. The serverless platform used in fog nodes is FaasD.[3] Java[4] was used with the Quarkus[5] framework to implement most modules as serverless functions. The *predictor* module was written in Python because it makes statistical predictions with built-in routines provided by *statsmodels*.[6] Healthcare services have an extra implementation for AWS Lambda due to entry-point differences compared with FaasD.

4.2. Infrastructure for assessment

AWS infrastructure was used to evaluate SmartVSO, while each fog node is introduced as an EC2 instance of type *t2.micro* with 1 vCPU and

1 https://github.com/GustavoASC/vital-signs-ingestion
2 https://python.org
3 https://github.com/openfaas/faasd
4 https://java.com
5 https://quarkus.io
6 https://statsmodels.org

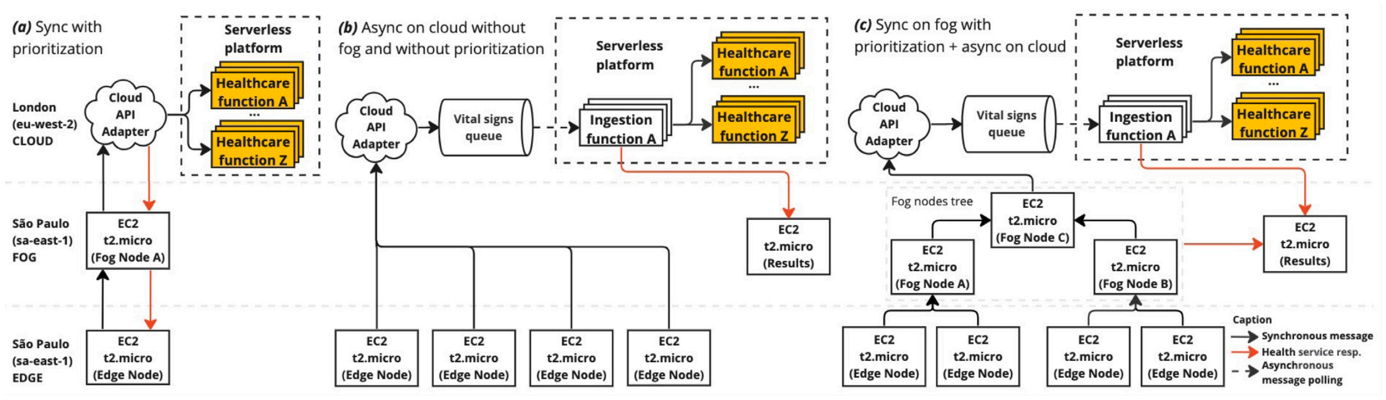

(a) Sync with prioritization
(b) Async on cloud without fog and without prioritization
(c) Sync on fog with prioritization + async on cloud
London (eu-west-2) CLOUD
São Paulo (sa-east-1) FOG
São Paulo (sa-east-1) EDGE
Cloud API Adapter
Serverless platform
Healthcare function A
Healthcare function Z
Vital signs queue
Ingestion function A
EC2 t2.micro (Fog Node A)
EC2 t2.micro (Fog Node B)
EC2 t2.micro (Fog Node C)
EC2 t2.micro (Edge Node)
EC2 t2.micro (Results)
Fog nodes tree
Caption
Synchronous message
Health service resp.
Asynchronous message polling

Fig. 9. Architectures evaluated in the experiments. In *(a)*, synchronous processing in the cloud and a single fog node. In *(b)*, an architecture without fog nodes, restricted to cloud without prioritization heuristics. In *(c)*, combining fog nodes with prioritization heuristics, and asynchronous processing in the cloud.

1GB RAM. Fog and edge nodes are located in São Paulo (sa-east-1), and resources in the cloud are located in London (eu-west-2). We use the term *edge nodes* to represent machines that artificially generate vital signs to fog nodes, which in a smart city is analogous to smartwatches sending vital signs to a fog node in the same neighborhood. Such cities were chosen to elucidate higher latency when offloading data to the cloud, instead of doing the entire experiment in São Paulo. The AWS API Gateway was used as the Cloud API Adapter and is connected to an SQS (Simple Queue Service) queue, so vital signs offloaded to the cloud can be processed asynchronously. AWS Lambda was the serverless platform chosen to run functions in the cloud because it integrates with the rest of AWS ecosystem. The architectures shown in Fig. 9 present how SmartVSO behaves from various perspectives.

Architecture A - synchronous processing on the cloud and a single fog node: this architecture processes vital signs synchronously (without queue) when reaching the cloud. Vital signs are generated from a single edge node and are forwarded to a single fog node. We used this architecture to develop and test the main components and emulate a small smart city. The Cloud API Adapter was implemented as a serverless function with AWS Lambda for this architecture, receiving vital signs and synchronously triggering the appropriate healthcare service as another function. Pros of this architecture include the ease of debugging and doing the first experiments, but this architecture cannot handle a large number of vital signs in a smart city.

Architecture B - asynchronous processing on the cloud without fog nodes: this architecture represents an approach composed of four edge nodes (representing several people in a smart city) sending vital signs directly to the queue on the cloud, without fog nodes. Compared to Architecture A, the main difference is that this architecture processes messages asynchronously and does not reject requests due to *throttling* errors. The queue may increase, but messages are consumed at a speed that serverless functions can ingest. Pros include handling a large number of vital signs. Cons include higher response times due to the physical distance between the edge nodes and the cloud, and no heuristic prioritizing urgent vital signs when consuming messages from this shared cloud queue.

Architecture C - asynchronous processing on the cloud with fog nodes: this is the most robust evaluated architecture because it combines fog computing with asynchronous processing in the cloud. This architecture considers a tree composed of two fog nodes at the first level and one fog node at the second level (the parent of the previous fog nodes). Four edge nodes are responsible for generating vital signs simultaneously for five user priorities. Pros include processing urgent vital signs with low response time in the fog due to the short physical distance with the source of vital signs, and the use of prioritization heuristics in the fog. Cons include not prioritizing messages in the cloud, also a limitation of previous architectures.

4.3. Evaluation metrics

The main goal of SmartVSO is to reduce response time for processing healthcare services with urgent vital signs. Our evaluation focuses on key metrics directly tied to this goal, providing insights into the model's effectiveness in prioritizing vital signs at scale. We expect a decrease in *response time (seconds)* for urgent vital signs, especially with high-priority healthcare services, as calculated rankings tend to be higher. An acceptable trade-off is an increase in response time for non-urgent vital signs, albeit prioritizing urgent ones. *Execution throughput (vital signs/sec)* measures the rate of vital signs processed per second for each user priority, accounting for the interval between processing the first and last vital sign, including queue waiting time in the cloud. *CPU load (percentage)* monitors computing resource usage, expecting CPU percentages on fog nodes to stay within lower and upper thresholds when triggering heuristics. *Waiting time in the queue (seconds)* is assessed to understand the queue's impact on response time, considered only in experiments using a cloud queue. Finally, with *offloading operations (count)* we expect high-priority vital signs to be primarily processed on leaf fog nodes, with non-urgent ones offloaded to higher fog levels or the cloud during resource overload.

4.4. Evaluation scenarios

This subsection details evaluations of the SmartVSO model under various scenarios. Vital signs were generated using virtual machines representing people in the neighborhood, given the controlled environment without individuals wearing smartwatches. The assessments involved including more fog nodes in the tree, vital sign quantities, and cloud processing strategies (synchronous or asynchronous). Notably, parameter selection for the experiments was challenging due to the vast matrix of possibilities. Initial thresholds were established and fine-tuned iteratively based on experiment results. Table 3 summarizes the experiments, while incoming paragraphs present them in detail.

Scenario 1 - Few vital signs to fog and cloud with Architecture A and a single healthcare service: this scenario involves vital signs representing a small city neighborhood with a small concentration of people. 1500 vital signs are considered, with 300 generated for each user priority and sent to a single fog node. Five vital signs are sent in parallel in the course of the experiment, one for each user priority. Evaluation happens with Architecture *A*, where vital signs are synchronously processed by spawning replicas of serverless functions on the cloud whenever a vital sign arises on this layer, or processing them in a single fog node if it has enough computing capabilities. Only the *body-temperature-monitor* healthcare service is considered.

Scenario 2 - Several vital signs to fog and cloud with Architecture A and a single healthcare service: this scenario considers a higher number of vital signs generated simultaneously, with the main

Table 3
Five main evaluation scenarios with specific workloads. Each scenario comprises one or more CPU-related configurations for the experiment. Healthcare service *B* stands for *body-temperature-monitor* and *H* stands for *heart-failure-predictor*. Some scenarios consider one healthcare service and others consider two.

Sce-	Vital signs		Parallel requests		Archi-	Health	Fog?	Cloud?	Async?	CPU config.		Offload thresh.	
nario	Total	Each pri.	Total	Each pri.	tecture	services				Strat.	Int.	Lower	Upper
#1a	1500	300	5	1	A	[B]	✓	✓	–	Naive	5s	75%	90%
#1b	1500	300	5	1	A	[B]	✓	✓	–	Naive	1s	75%	90%
#1c	1500	300	5	1	A	[B]	✓	✓	–	Aging	1s	75%	90%
#2	4500	900	15	3	A	[B]	✓	✓	–	Aging	1s	75%	90%
#3	80,000	16,000	80	16	B	[B, H]	–	✓	✓	–	–	–	–
#4	40,000	8000	40	8	C	[H]	✓	✓	✓	Aging	1s	60%	98%
#5a	80,000	16,000	80	16	C	[B, H]	✓	✓	✓	Aging	1s	60%	98%
#5b	80,000	16,000	80	16	C	[B, H]	✓	✓	✓	Aging	1s	80%	98%

goal of understanding how Architecture A deals with the increase in vital signs. Considering 4500 vital signs, 900 are generated per user priority, with three vital signs sent concurrently for each priority. This scenario also considers the *body-temperature-monitor* healthcare service and is done with the *aging* technique for collecting CPU observations, while considering offloading thresholds of 75% and 90%.

Scenario 3 - Many vital signs only to the cloud with Architecture B and two healthcare services: this scenario considers Architecture B, which does not have fog nodes to process vital signs. All vital signs are artificially generated and sent directly to the cloud. This scenario is important because it serves as a reference to which we can compare the results with scenarios that employ fog nodes. 80,000 vital signs are considered, while 16,000 are generated for each user priority, and 16 vital signs are sent in parallel for each priority. This results in 80 vital signs sent simultaneously. This scenario considers *body-temperature-monitor* and *heart-failure-predictor* healthcare services, each processing half of the vital signs (40,000).

Scenario 4 - Many vital signs to fog and cloud with Architecture C and a single healthcare service: this scenario utilizes Architecture *C*, with 8000 vital signs for each of the five user priorities and 8 vital signs simultaneously generated for each priority. Resulting in 40,000 vital signs, 40 are sent simultaneously to the fog. With two fog nodes at the first hierarchy level, each receives half of the vital signs. Across four edge nodes, each sends 10,000 vital signs, matching the overall count. Only the *heart-failure-predictor* service is considered to assess the Ranking Heuristic's behavior. This choice is based on its heavier computational load, stressing the CPU, compared to *body-temperature-monitor*. The scenario, combining different priorities and a single service, makes offloading decisions either by exceeding the upper CPU threshold or activating the Ranking Heuristic, excluding the Duration Heuristic as there is only one healthcare service involved.

Scenario 5 - Many vital signs to fog and cloud with two services and Architecture C: this is the most robust evaluation scenario, involving 80,000 vital signs, multiple fog nodes, and asynchronous cloud processing. Unlike the previous scenario, it incorporates two healthcare services (*body-temperature-monitor* and *heart-failure-predictor*). The vital sign count is doubled to 80,000, with each service consuming half the messages. For each user priority, 16,000 vital signs are sent, and 16 parallel requests simultaneously transmit vital signs, resulting in 80,000 vital signs when considering all user priorities. Ranking collisions are expected, as multiple vital signs share the same calculated ranking. The Duration Heuristic then determines the offloading based on predictions of service completion duration.

5. Results

This section presents results for the evaluated scenarios with the methodology presented in Section 4. Detailed information about response time, execution throughput, CPU load, waiting time in the queue, and offloading operations are presented.

5.1. Scenario 1 - Few vital signs to fog and cloud with Architecture A and a single healthcare service

Fig. 10 compiles results for the first scenario, encompassing synchronous processing of a small number of vital signs (1500) with fog and cloud using Architecture A. We present three results because we did three experiments with the same architecture and the same number of vital signs, but different parameters that influence how the CPU utilization is collected.

In Fig. 10*(a)*, CPU utilization is collected with a single thread and a 5-second interval between each collection. The throughput of vital signs processed per second for each user priority is suboptimal. We see high-priority vital signs not being appropriately favored, attributed to the Ranking Heuristic not being triggered, as the CPU utilization is either below or above lower and upper thresholds, respectively. The offloading operations follow a consistent pattern across priorities, processing approximately 250 vital signs locally and offloading 50 vital signs for each priority. Response times remain consistent across priorities for the same reason. In the response time chart, initial values are higher due to the serverless function cold start, a one-time occurrence. Processing vital signs in the fog (São Paulo) takes around 150 ms, while processing them in the cloud (London) requires around 400 ms due to the greater physical distance. Regarding CPU utilization on the fog node, we see a repeating and undesired behavior leading to the system making decisions based on outdated values. This happens because CPU utilization is not detected quickly enough. Heuristics would only be triggered for this experiment when CPU utilization is between 75% and 90%, but are not triggered as the utilization is either lower or higher than these values.

Fig. 10*(b)* presents results for the second experiment considering a 1-second time window to collect CPU utilization. The previous experiment used a larger time window of 5 s, so reducing this interval is an effort to provide CPU observations to the offloading algorithm more promptly, enhancing the responsiveness to processing changes. Still, CPU observations continue to range from extremely low to extremely high values, such as close to 20% and 100%, respectively. Charts in this experiment show a similar pattern to the previous one, so we will not provide detailed explanations for each chart in this second experiment. The charts for CPU utilization and response time exhibit more notable behavior, sharing the same format as the previous experiment but with increased frequency due to a shorter CPU collection time window. Simply reducing this interval is insufficient to enhance SmartVSO's quality of experience, as heuristics were triggered only a few times.

Finally, Fig. 10*(c)* presents results for the third experiment, incorporating the smoothing technique for more stable CPU observations [32]. The CPU usage chart has two lines — yellow for raw, unsmoothed CPU usage, reflecting the traditional approach, and blue for smoothed CPU utilization using the aging technique, considering the last six observations. Offloading operations no longer have a drastic impact on the observed CPU, resulting in smoother decision-making. Throughput of vital signs processed per second is similar between priorities, as

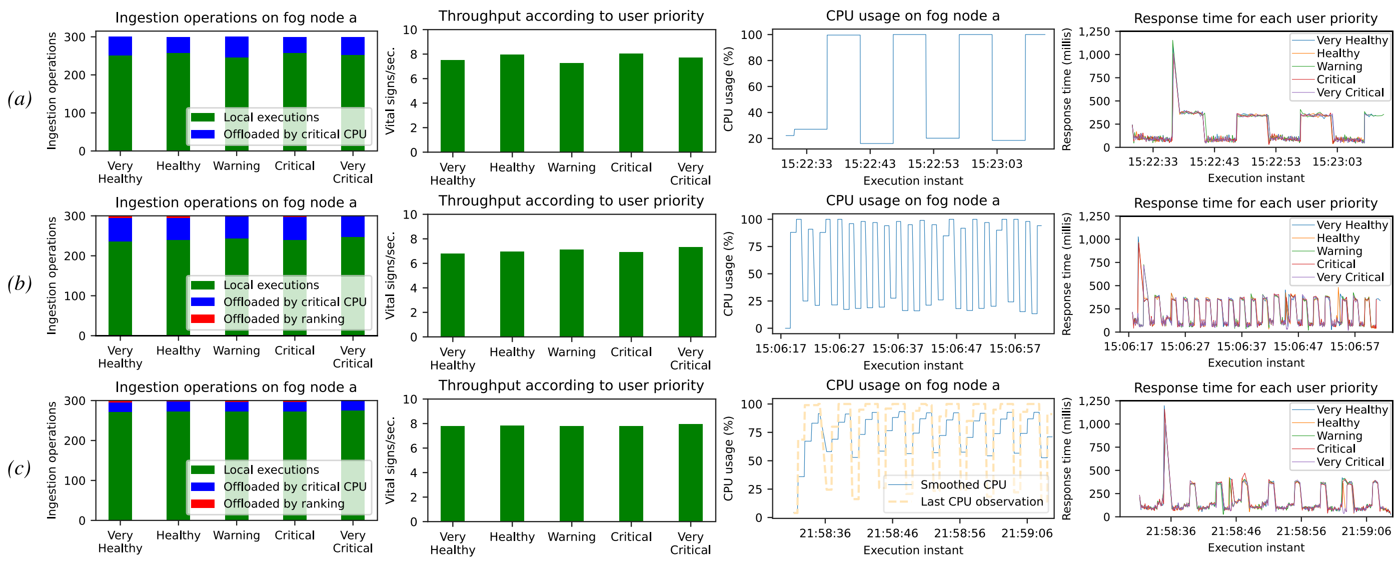


Fig. 10. Results for Scenario 1 comprising three CPU configurations. In *(a)*, we consider a naive CPU collection strategy with CPU collection every 5 s. In *(b)*, we reduce this interval to 1 s. In *(c)*, we consider the Aging technique to smooth the CPU usage based on previous observations.

most vital signs were processed locally regardless of the priority. Most offloading occurred because the smoothed CPU exceeded the upper threshold, with only a small fraction due to the Ranking Heuristic. The *very critical* priority only experienced offloading when exceeding the upper threshold.

In conclusion, the first scenario highlights that selecting the CPU utilization collection strategy and defining offloading parameters significantly influences SmartVSO's performance. Incorrect definitions may lead to a lack of prioritization for vital signs from individuals in urgent situations, as identified CPU utilization can exhibit sudden peaks beyond pre-defined thresholds. Additionally, we recognize that 1500 vital signs represent a limited dataset, and increasing this number is crucial for a comprehensive understanding of the model's behavior, which is the focus of the second scenario.

5.2. Scenario 2 - Several vital signs to fog and cloud with Architecture A and a single healthcare service

In the second scenario, we increase the number of vital signs and continue using Architecture A, which processes vital signs synchronously on both the fog and cloud layers. This experiment resulted in discarding a subset of vital signs due to *throttling* errors on AWS Lambda. The AWS account's default maximum limit of 10 concurrent Lambda executions was exceeded. Though this limit can be increased, it poses a potential obstacle for employing Architecture *A* in smart cities, where the number of people in neighborhoods fluctuates during the day. In this experiment, 809 out of 4500 vital signs failed to be processed, while 3691 vital signs were successfully processed across fog and cloud. During the experiment, only up to 10 replicas of the *body-temperature-monitor* healthcare serverless function were allowed simultaneously. Any additional offloading operations beyond this limit of 10 replicas would fail, resulting in the discarding of those offloaded vital signs. Executions take longer (approximately 3 s) during *throttling* errors, as the HTTP connection remains established with the cloud provider until the *throttling* error is identified and a response is returned. In conclusion, synchronous processing of vital signs in fog and cloud layers is unsuitable for the healthcare domain in smart cities. Limited computing resources may fail to handle vital signs from concentrated neighborhoods, resulting in lost data during cloud resource overload. Storing incoming vital signs in a cloud queue and processing them asynchronously when cloud resources are available is vital, as discussed in Scenario 3.

5.3. Scenario 3 - Many vital signs only to the cloud with Architecture B and two healthcare services

A larger volume of vital signs is processed in this scenario. Data is directly sent to a cloud queue and asynchronously processed by serverless functions to prevent *throttling* errors. Notably, fog nodes and heuristics for prioritizing vital signs from people in urgent situations are not considered. In Fig. 11, two charts depict the response time percentiles and throughput for each user priority in Scenario 3. Both metrics show similarity among priorities, indicating that urgent vital signs are not given preference due to the absence of prioritization heuristics in the cloud. This experiment clarifies that a cloud queue effectively handles the influx of vital signs from specific neighborhoods. Despite delayed processing for vital signs at the end of the queue, none are lost due to *throttling* errors during cloud resource overload, as observed in the previous scenario. This underscores the efficiency of using a queue for asynchronous vital sign processing in the healthcare domain. While existing works may not directly integrate healthcare with serverless computing, utilizing a queue remains crucial in preventing cloud resource overload during sudden usage peaks. Works like [24,25] do not explicitly mention queue usage in their cloud solutions.

5.4. Scenario 4 - Many vital signs to fog and cloud with Architecture C and a single healthcare service

This scenario combines synchronous processing of vital signs on fog nodes with asynchronous processing in the cloud. This design prioritizes urgent vital signs and enables SmartVSO to handle heavy workloads. Fig. 11 compiles the results from this experiment, while the throughput chart may initially suggest that the architecture does not prioritize urgent vital signs. However, in reality, it favors urgent vital signs because smaller percentiles show a smaller response time for them.

This figure also details offloading operations on fog nodes and the number of vital signs processed locally. The Ranking Heuristic results in more offloading for healthier individuals, as expected. This underscores the effectiveness of prioritization in the proposed fog-cloud healthcare architecture, a feature absent in some related works like [21,25,26]. Fog nodes *a* and *b* exhibit similar offloading behavior due to receiving the same number of vital signs from edge nodes (representing individuals in a smart city). For *very critical* user priority, vital signs are primarily processed locally on fog nodes *a* and *b* and

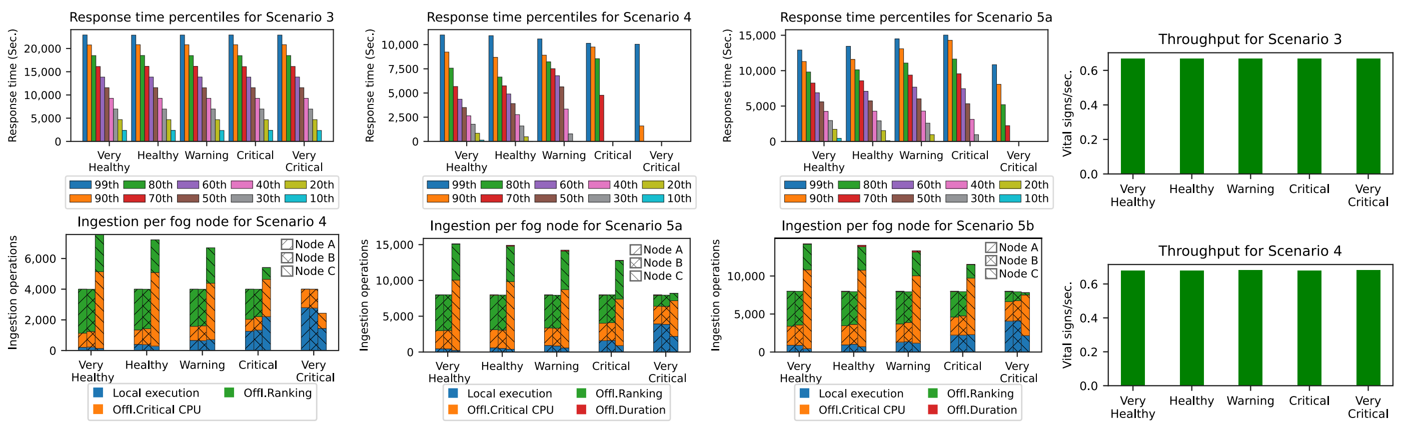


Fig. 11. Results for Scenarios 3, 4, and 5. Response time percentiles indicate how long it takes to process vital signs for each user priority, considering the moment the vital signs were generated, until they were processed, either in the fog or in the cloud. The ingestion operations charts indicate how many vital signs were processed locally in each fog node, and when offloaded, the reason why they were offloaded. Throughput charts for Scenarios 3 and 4 are also presented.

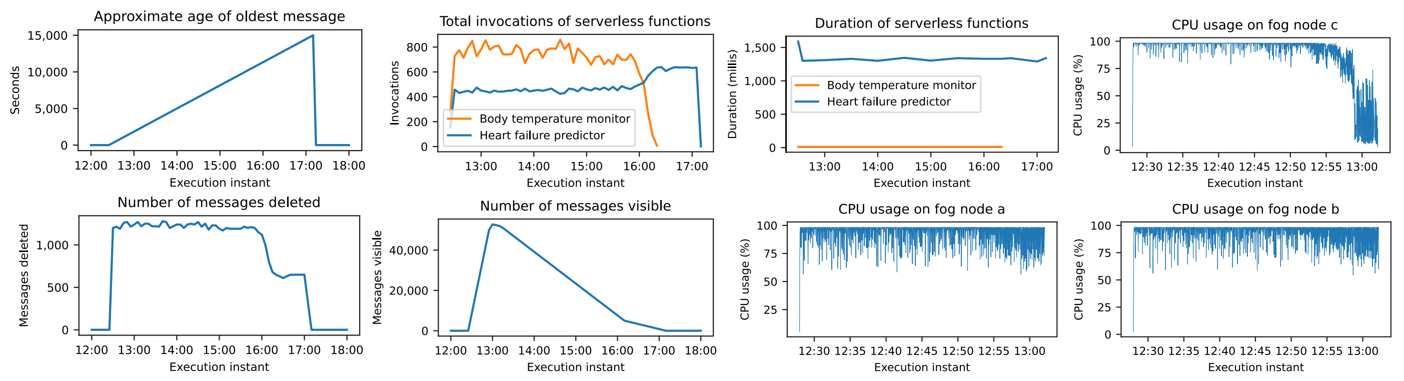


Fig. 12. Results for Scenario 5 considering 60% as the lower CPU threshold. 80.000 vital signs were generated and processed with two healthcare services and Architecture C. Charts present utilization of the queue in the cloud, metrics about the healthcare serverless functions in the cloud, and CPU utilization on fog nodes.

never offloaded due to the Ranking Heuristic. The *very healthy* priority has around 200 vital signs processed locally on these fog nodes, while *very critical* priority has around 3000 processed locally. Offloading due to CPU usage exceeding the upper threshold is similar for every user priority, indicating fog node resource overload. Since this experiment simultaneously sends the same number of vital signs for each user priority, it is expected that the number of offloading operations by exceeding the upper threshold is also similar for each user priority. Notably, the Ranking Heuristic leads to more offloading for *very healthy* priority. The figure also depicts fog nodes *a* and *b* receiving 20,000 vital signs in total, with 4000 for each priority.

Fig. 11 also illustrates how fog node *c* manages offloaded vital signs from fog nodes *a* and *b*. The offloading pattern on fog node *c* differs from other fog nodes, with most vital signs offloaded due to CPU usage exceeding the upper threshold, not the Ranking Heuristic. This difference is because fog node *c* acts as the parent of two fog nodes, receiving more vital signs simultaneously. Except for *very critical* user priority, fog node *c* received over 4000 vital signs for each priority, representing a higher volume than first-level fog nodes.

In conclusion, Architecture C is promising because combining heuristics on fog nodes with asynchronous processing in the cloud leads to better results. Offloading heuristics in the fog prioritize vital signs for urgent situations, while asynchronous processing in the cloud manages peak loads when people concentrate in specific neighborhoods, addressing limitations of previous experiments. Scenario 5, featuring increased vital signs and two healthcare services, will be presented next.

5.5. Scenario 5 - Many vital signs to fog and cloud with two services and Architecture C

This scenario includes two experiments with different offloading thresholds: the first with thresholds set at 60% and 98% for lower and upper thresholds, and the second at 80% and 98%. In Fig. 11, response time percentiles is shown for the first experiment *5a*. Individuals prioritized as *very critical* experience shorter response times, showcasing the effectiveness of the Ranking Heuristic. This efficiency is more evident at lower percentiles but is still noticeable at higher percentiles. For *very critical* conditions, response times range between 3 and 5 s at the 50th and 60th percentiles, respectively. In contrast, for *very healthy* individuals, the range spans from 5800 s (97 min) to 7000 s (116 min). This can be decisive in saving a person's life. In the same chart, a notable distinction emerges between the 60th and 70th percentiles for individuals in *very critical* conditions. Processing vital signs for the 70th percentile can take up to 2235 s (37 min) due to offloading a subset of vital signs at the end of a heavily utilized, shared queue in the cloud. Despite this, the 70th percentile for *very critical* priority remains smaller than other priority levels. For instance, *critical* priority requires up to 9561 s (159 min) to process 70% of vital signs. The response time disparity is significant, especially for the *very critical* priority, with no visible value in the 60th percentile. This indicates almost instantaneous processing in such a subset of vital signs.

Another chart in Fig. 11 shows offloading operations on each fog node for Scenario 5a. Nodes *a* and *b* exhibit similar behavior, given their identical computing resources and vital sign intake. As expected, the offloading strategy aligns with priority levels: more critical priorities experience increased local executions and reduced offloading operations, emphasizing prompt processing for urgent conditions. This underscores the effectiveness of the tree-based architecture, where the parent fog node supports local vital sign processing before offloading to the cloud — a feature absent in some related works like [22,23,25]. However, the impact of the Duration Heuristic on offloading operations in this experiment is minimal. Moreover, offloading operations due to CPU utilization exceeding the upper threshold is consistent across user priorities, reflecting the absence of ranking considerations in this overloaded computing resource scenario. Conversely, fog node *c*, serving as the parent of fog nodes *a* and *b*, exhibits a different offloading pattern. As it receives vital signs simultaneously from both child nodes, the total offloading operations are higher on fog node *c*, with minimal local processing — a behavior indicating that an overloaded parent requires more computing resources than its child nodes.

In Fig. 12, the first chart is the approximate age of the oldest message in the queue on the cloud. This information represents the time it took for the oldest vital sign to be processed by a healthcare serverless function. In this experiment, the maximum waiting time reached around 15,100 s (4 h and 11 min). This poses a challenge for individuals with *very critical* priority, necessitating rapid responses. Offloading more vital signs to the cloud queue extends the time for asynchronous processing. The figure also includes a chart illustrating message deletions from the queue within 5-minute intervals. A consistent deletion pattern is observed until 16:00, averaging around 1200 to 1300 deletions every 5 min. After 16:00, fewer vital signs are deleted, indicating a concentration of processing by the *heart-failure-predictor* service, known for its heavier workload and longer processing times.

Additionally, this figure includes a chart depicting the number of visible messages in the queue within a 5-minute window. Vital signs are added to the queue between 12:30 and 13:00, the interval when vital signs are generated. Vital signs not processed on the fog are offloaded to the cloud during this period, starting processing upon arrival in the queue. However, the number of generated vital signs surpasses the parallel consumption capacity, resulting in a visible message count decrease only after 13:00 when vital sign generation ceases, yet processing by cloud serverless functions persists.

The next two charts in Fig. 12 display the total invocations of healthcare services (functions) in the cloud and their respective durations within a 5-minute time frame. The *body-temperature-monitor* function completes fast and has a higher invocation frequency (800–850 times every five minutes), while the *heart-failure-predictor* function, taking longer (around 1.5 s), is invoked around 430 times in the same interval. These values correspond to the number of messages deleted from the queue, as messages are automatically consumed and deleted upon successful processing by a serverless function. Post-16:00, the total invocations of *heart-failure-predictor* rise due to the absence of messages related to *body-temperature-monitor*, as each message is assigned to a healthcare service.

Finally, in Fig. 12, CPU utilization on each fog node is depicted during the initial 30 min of the experiment when vital signs are generated. Fog nodes *a* and *b* show CPU utilization between 60% and 100%, being on the first level of the tree and receiving an equal number of vital signs. Fog node *c* consistently maintains CPU close to 100% since it acts as the parent node and concurrently receives vital signs offloaded by others. Notably, fog node *c* experiences a decrease in CPU usage as vital sign generation concludes, resulting in more available CPU due to fewer offloading operations from fog nodes *a* and *b*.

In the second experiment of Scenario 5 (5b), we raised the lower CPU threshold from 60% to 80% to increase local processing of vital signs. The previous setup resulted in many vital signs offloaded to fog node *c* (the tree's root) and the cloud, causing higher response times. Fig. 11 illustrates the impact of the increased lower threshold on fog layer offloading operations. Specifically, for the *very healthy* user priority, we observe a nearly twofold increase in local executions on

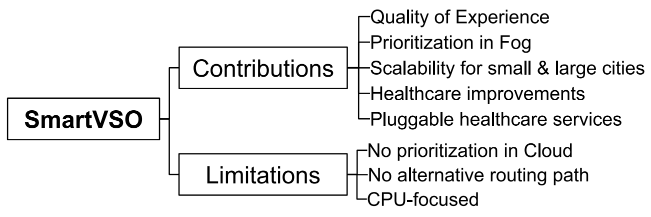

SmartVSO
Contributions
Quality of Experience
Prioritization in Fog
Scalability for small & large cities
Healthcare improvements
Pluggable healthcare services
Limitations
No prioritization in Cloud
No alternative routing path
CPU-focused

Fig. 13. Main contributions and limitations of SmartVSO model.

fog node *a*, rising from 451 to around 1000. However, there is minimal difference for the *very critical* user priority. Fog node *c* shows increased local executions, particularly for user priorities from *very healthy* to *critical*, with no significant change for *very critical* user priority. Notably, most offloading operations for *very critical* user priority still occur due to surpassing the upper CPU threshold on this root fog node.

6. Discussion

This section discusses the results presented in Section 5 and emphasizes the proposed model's contributions and limitations, with Fig. 13 summarizing these topics.

6.1. Contributions

SmartVSO improves the quality of experience in smart cities by processing vital signs with reduced response time for people in urgent conditions. This helps improve health in several manners, such as identifying if a person is having heart failure, knowing which neighborhoods have more problems with a specific disease, and calling an ambulance automatically. The Ranking Heuristic contributes to this and relies on a simple calculation that requires minimal computational effort and generates significant outcomes. In the context of Scenario 5 with 60% and 98% for CPU thresholds, 60% of vital signs from individuals with *very critical* priority are processed in less than 5.3 s. It ranges from 6873 s (114 min) to 7678 s (127 min) to process this same percentage of vital signs for other user priorities.

Fog nodes arranged in tree topology combined with a queue in the cloud contribute to scalability, making the architecture suitable for both smaller and larger cities. Experiments with Architecture A evidence shorter response times when processing vital signs in the fog because of its shorter physical distance (150 ms), while processing in the cloud in London takes around 400 ms. The impact of prioritization heuristics is visible when comparing architectures B and C. The former only consumes messages from the cloud without prioritization techniques, resulting in similar response times for all priorities. The latter leads to reduced response times for individuals in urgent situations because they are mostly processed in the fog. Focusing on the 60% percentile for people with *very critical* priority, in Architecture C, vital signs are processed in no more than 5.3 s, while in Architecture B it takes around 14,000 s for this same percentile. Possibilities to improve the throughput of SmartVSO include adding more fog nodes to enable processing of more vital signs locally, and increasing the quotas for serverless functions on the cloud provider to process more healthcare services simultaneously.

Finally, SmartVSO's performance is directly impacted by offloading strategy and thresholds, but determining optimal values is challenging. We understand that no specific threshold fits all scenarios and could be dynamically calibrated. Setting the lower threshold too small causes premature offloading of non-critical vital signs because the Ranking Heuristic would be triggered too often, but the fog node would have more computational resources to process critical vital signs locally. On the other hand, during subtle concentrations of people in specific neighborhoods due to social events, it is prudent to reduce the lower threshold to pre-allocate resources for people in urgent conditions, thus offloading non-critical vital signs promptly.

6.2. Limitations

The absence of a prioritization strategy for vital signs in the cloud results in a shared queue, which includes urgent cases offloaded by the last fog node. Currently, SmartVSO relies solely on CPU usage for offloading decisions, limiting its considerations. Future enhancements could incorporate additional metrics such as memory, network bandwidth, and latency between fog nodes and the cloud. Further investigation into the Duration Heuristic is warranted, given its limited offloading of vital signs. Subsequent experiments may explore combining thresholds and workloads to trigger this heuristic more effectively, especially for CPU-intensive healthcare services processing vital signs of the same priority. In terms of scalability, an improvement could involve considering communication between sibling fog nodes when only a specific branch of the fog tree is overloaded, rather than relying on vertical offloading toward the cloud. The current architecture also faces challenges when communication issues occur with the parent fog node, as there is a single path for vital sign routing. On the security front, the model has limitations. It lacks an encryption module for data transiting between fog nodes and the cloud. This is a critical concern for preserving user privacy, especially since vital signs are sensitive data. Furthermore, the model cannot detect malicious changes in the data, which is essential for identifying falsified vital signs. These improvements would enhance the overall security posture of SmartVSO.

7. Conclusion

Vital signs monitoring can be decisive in saving a person's life, which is promising in smart cities. Healthcare services can improve the health quality of the population by triggering alerts whenever a problem is identified from the collected vital signs. This paper presented SmartVSO, a computational model for ingesting vital signs in a fog-cloud environment with offloading strategies to reduce response time for critical vital signs and deal with usage peaks by leveraging synchronous processing in the fog and asynchronous processing in the cloud, respectively. In this scalable model, serverless computing allows authorized companies to easily implement and deploy healthcare services while focusing on the healthcare domain instead of dealing with scalability and placement decisions, which is the responsibility of the proposed model. Results are exciting and indicate the effectiveness of the prioritization heuristics, and we are analyzing the possibility of implementing this in a smart city in the south of Brazil. As future work, we suggest *(i)* employ prioritization heuristics when consuming messages from the queue in the cloud, *(ii)* introduce algorithms to calibrate lower and upper offloading thresholds, *(iii)* use different workloads that trigger the duration heuristic more frequently, such as concentrating the generation of vital signs for a specific user priority, and *(iv)* consider extra metrics in the offloading decision in addition to the CPU, such as network latency and memory.

CRediT authorship contribution statement

Gustavo André Setti Cassel: Conceptualization, Data curation, Investigation, Methodology, Resources, Software, Validation, Writing – original draft, Writing – review & editing. **Rodrigo da Rosa Righi:** Conceptualization, Investigation, Supervision, Writing – original draft, Writing – review & editing. **Cristiano André da Costa:** Conceptualization, Investigation, Methodology, Writing – original draft, Writing – review & editing. **Marta Rosecler Bez:** Conceptualization, Investigation, Supervision, Validation. **Marcelo Pasin:** Data curation, Investigation, Validation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Rodrigo da Rosa Righi reports was provided by Foundation for Research Support of Rio Grande do Sul State. Rodrigo da Rosa Righi and Cristiano Costa reports was provided by National Council for Scientific and Technological Development.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the following Brazilian agencies: FAPERGS (processes 19/2551-0001340-0, 21/2551-0000118-6 and 23/2551-0002202-8), CAPES (financial code 001), and CNPq (grant numbers 309537/2020-7, 305263/2021-8 and 408524/2023-5).

References

[1] A. Rejeb, K. Rejeb, H. Treiblmaier, A. Appolloni, S. Alghamdi, Y. Alhasawi, M. Iranmanesh, The internet of things (IoT) in healthcare: Taking stock and moving forward, Internet Things 22 (2023) 100721, http://dx.doi.org/10.1016/j.iot.2023.100721.

[2] V.F. Rodrigues, R. da Rosa Righi, C.A. da Costa, F.A. Zeiser, B. Eskofier, A. Maier, D. Kim, Digital health in smart cities: Rethinking the remote health monitoring architecture on combining edge, fog, and cloud, Health Technol. (2023) http://dx.doi.org/10.1007/s12553-023-00753-3.

[3] M. Kumar, A. Kumar, S. Verma, P. Bhattacharya, D. Ghimire, S.-h. Kim, A.S.M.S. Hosen, Healthcare internet of things (H-IoT): Current trends, future prospects, applications, challenges, and security issues, Electronics 12 (9) (2023) http://dx.doi.org/10.3390/electronics12092050.

[4] V.A. Dang, Q. Vu Khanh, V.-H. Nguyen, T. Nguyen, D.C. Nguyen, Intelligent healthcare: Integration of emerging technologies and internet of things for humanity, Sensors 23 (9) (2023) http://dx.doi.org/10.3390/s23094200.

[5] M. Ezhilarasi, A. Kumar, M. Shanmugapriya, A. Ghanshala, A. Gupta, Integrated healthcare monitoring system using Wireless Body Area networks and internet of things, in: 2023 4th International Conference on Innovative Trends in Information Technology, ICITIIT, 2023, pp. 1–5, http://dx.doi.org/10.1109/ICITIIT57246.2023.10068616.

[6] O. Alfandi, An intelligent IoT monitoring and prediction system for health critical conditions, Mob. Netw. Appl. 27 (3) (2022) 1299–1310, http://dx.doi.org/10.1007/s11036-021-01892-5.

[7] S. Krishnamoorthy, A. Dua, S. Gupta, Role of emerging technologies in future IoT-driven healthcare 4.0 technologies: A survey, current challenges and future directions, J. Ambient Intell. Humaniz. Comput. 14 (1) (2023) 361–407, http://dx.doi.org/10.1007/s12652-021-03302-w.

[8] H.F. Ahmad, W. Rafique, R.U. Rasool, A. Alhumam, Z. Anwar, J. Qadir, Leveraging 6G, extended reality, and IoT big data analytics for healthcare: A review, Comp. Sci. Rev. 48 (2023) 100558, http://dx.doi.org/10.1016/j.cosrev.2023.100558.

[9] A. Alsiddiky, W. Awwad, K. bakarman, H. Fouad, A.S. Hassanein, A.M. Soliman, Priority-based data transmission using selective decision modes in wearable sensor based healthcare applications, Comput. Commun. 160 (2020) 43–51, http://dx.doi.org/10.1016/j.comcom.2020.05.039.

[10] G.A.S. Cassel, V.F. Rodrigues, R. da Rosa Righi, M.R. Bez, A.C. Nepomuceno, C. André da Costa, Serverless computing for internet of things: A systematic literature review, Future Gener. Comput. Syst. 128 (2022) 299–316, http://dx.doi.org/10.1016/j.future.2021.10.020.

[11] B. Cheng, J. Fuerst, G. Solmaz, T. Sanada, Fog function: Serverless fog computing for data intensive IoT services, in: 2019 IEEE International Conference on Services Computing, SCC, 2019, pp. 28–35, http://dx.doi.org/10.1109/SCC.2019.00018.

[12] Z. Rezazadeh, M. Rezaei, M. Nickray, LAMP: A hybrid fog-cloud latency-aware module placement algorithm for IoT applications, in: 2019 5th Conference on Knowledge Based Engineering and Innovation, KBEI, 2019, pp. 845–850, http://dx.doi.org/10.1109/KBEI.2019.8734958.

[13] D. Bermbach, S. Maghsudi, J. Hasenburg, T. Pfandzelter, Towards auction-based function placement in serverless fog platforms, in: 2020 IEEE International Conference on Fog Computing, ICFC, 2020, pp. 25–31, http://dx.doi.org/10.1109/ICFC49376.2020.00012.

[14] G. George, F. Bakir, R. Wolski, C. Krintz, NanoLambda: Implementing functions as a service at all resource scales for the internet of things, in: 2020 IEEE/ACM Symposium on Edge Computing, SEC, 2020, pp. 220–231, http://dx.doi.org/10.1109/SEC50012.2020.00035.
[15] H.O. Hassan, S. Azizi, M. Shojafar, Priority, network and energy-aware placement of IoT-based application services in fog-cloud environments, IET Commun. 14 (13) (2020) 2117–2129, http://dx.doi.org/10.1049/iet-com.2020.0007.
[16] I. Pelle, F. Paolucci, B. Sonkoly, F. Cugini, Latency-sensitive edge/cloud serverless dynamic deployment over telemetry-based packet-optical network, IEEE J. Sel. Areas Commun. 39 (9) (2021) 2849–2863, http://dx.doi.org/10.1109/JSAC.2021.3064655.
[17] C.K. Dehury, S. Poojara, S.N. Srirama, Def-DReL: Towards a sustainable serverless functions deployment strategy for fog-cloud environments using deep reinforcement learning, Appl. Soft Comput. 152 (2024) 111179, http://dx.doi.org/10.1016/j.asoc.2023.111179.
[18] T. Rausch, A. Rashed, S. Dustdar, Optimized container scheduling for data-intensive serverless edge computing, Future Gener. Comput. Syst. 114 (2021) 259–271, http://dx.doi.org/10.1016/j.future.2020.07.017.
[19] C. Cicconetti, M. Conti, A. Passarella, A decentralized framework for serverless edge computing in the internet of things, IEEE Trans. Netw. Serv. Manag. 18 (2) (2021) 2166–2180, http://dx.doi.org/10.1109/TNSM.2020.3023305.
[20] A. AlZailaa, H.R. Chi, A. Radwan, R. Aguiar, Low-latency task classification and scheduling in fog/cloud based critical e-health applications, in: ICC 2021 - IEEE International Conference on Communications, 2021, pp. 1–6, http://dx.doi.org/10.1109/ICC42927.2021.9500985.
[21] U. Arora, N. Singh, IoT application modules placement in heterogeneous fog–cloud infrastructure, Int. J. Inf. Technol. 13 (5) (2021) 1975–1982, http://dx.doi.org/10.1007/s41870-021-00672-4.
[22] M.M. Bukhari, T.M. Ghazal, S. Abbas, M.A. Khan, U. Farooq, H. Wahbah, M. Ahmad, K.M. Adnan, An intelligent proposed model for task offloading in fog-cloud collaboration using logistics regression, Comput. Intell. Neurosci. 2022 (2022) 3606068, http://dx.doi.org/10.1155/2022/3606068.
[23] Y.-A. Daraghmi, E.Y. Daraghmi, R. Daraghma, H. Fouchal, M. Ayaida, Edge–fog–cloud computing hierarchy for improving performance and security of NB-IoT-Based health monitoring systems, Sensors 22 (22) (2022) http://dx.doi.org/10.3390/s22228646.
[24] M. Hajvali, S. Adabi, A. Rezaee, M. Hosseinzadeh, Software architecture for IoT-based health-care systems with cloud/fog service model, Cluster Comput. 25 (1) (2022) 91–118, http://dx.doi.org/10.1007/s10586-021-03375-4.
[25] A. Gupta, V.K. Chaurasiya, Efficient task-offloading in IoT-fog based health monitoring system, in: 2022 OITS International Conference on Information Technology, OCIT, 2022, pp. 495–500, http://dx.doi.org/10.1109/OCIT56763.2022.00098.
[26] A.M. Jasim, H. Al-Raweshidy, Towards a cooperative hierarchical healthcare architecture using the HMAN offloading scenarios and SRT calculation algorithm, IET Networks 12 (1) (2023) 9–26, http://dx.doi.org/10.1049/ntw2.12064.
[27] W.T. Vambe, Fog computing quality of experience: Review and open challenges, Int. J. Fog Comput. (IJFC) 6 (1) (2023) 1–16, http://dx.doi.org/10.4018/IJFC.317110.
[28] M. Varela, L. Skorin-Kapov, T. Ebrahimi, Quality of service versus quality of experience, in: S. Möller, A. Raake (Eds.), Quality of Experience: Advanced Concepts, Applications and Methods, Springer International Publishing, Cham, 2014, pp. 85–96, http://dx.doi.org/10.1007/978-3-319-02681-7_6.
[29] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, O'Reilly Media, 2017.
[30] H. Simon Junior, C. Schvartsman, G.d.A. Sukys, S.C.L. Farhat, Pediatric emergency triage systems, Rev. Paul Pediatr. 41 (2022) e2021038.
[31] R. Hyndman, G. Athanasopoulos, Forecasting: Principles and Practice, third ed., OTexts, Australia, 2021.
[32] R.d.R. Righi, V.F. Rodrigues, C.A. da Costa, G. Galante, L.C.E. de Bona, T. Ferreto, AutoElastic: Automatic resource elasticity for high performance applications in the cloud, IEEE Trans. Cloud Comput. 4 (1) (2016) 6–19, http://dx.doi.org/10.1109/TCC.2015.2424876.

Gustavo Cassel received his Master degree in Applied Computing from Unisinos University. His B.Sc. degree in Computer Science was obtained in 2020, from Feevale University. He is currently pursuing his M.Sc. degree in Applied Computing in University of Vale do Rio dos Sinos, Brazil. His research interests include: elasticity, distributed applications, cloud computing and Function as a Service.

Rodrigo da Rosa Righi is assistant professor and researcher at University of Vale do Rio dos Sinos, Brazil, where he advises master and Ph.D. students. Rodrigo concluded his postdoctoral studies at KAIST — Korea Advanced Institute of Science and Technology, under the following topics: RFID and cloud computing. He obtained his Ph.D. degree in Computer Science from the UFRGS University, Brazil, in 2005. His research interests include load balancing and process migration. Finally, he is a senior member of the IEEE and ACM.

Cristiano André da Costa is a full professor at the Universidade do Vale do Rio dos Sinos, Brazil, and a researcher on productivity at CNPq (National Council for Scientific and Technological Development). He received the Ph.D. degree in computer science from UFRGS University, Brazil, in 2008. His research interests include ubiquitous, mobile, parallel, and distributed computing. He is a member of the ACM, IEEE, IADIS, and the Brazilian Computer Society.

Marta Rosecler Bez obtained her Ph.D. in Informatics in Education from the Federal University of Rio Grande do Sul (2013) and M.Sc. in Computer Science from the Pontifical Catholic University of Rio Grande do Sul (2001). She is currently a professor in the Professional Master's in Creative Industry at Feevale University.

Marcelo Pasin is researcher in the University of Neuch tel (Switzerland) and an associate professor in the Engineering School Arc of the University of Applied Sciences and Arts Western Switzerland. I hold diplomas of Doctor in Computer Science from the National Polytechnic Institute of Grenoble (France, 1999), Master in Computer Science from the Federal University of Rio Grande do Sul (Porto Alegre, Brazil, 1994) and Electrical Engineering from the Federal University of Santa Maria (Brazil, 1988). After graduating, I worked two years for the computer industry in Brazil. Following that, I was tenured as assistant and later associate professor at the Federal University of Santa Maria (Brazil, 1991–2007). He is member of IEEE, ACM and SBC (Brazil).