Simulation Modelling Practice and Theory 111 (2021) 102336

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

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

A scheduling-based dynamic fog computing framework for augmenting resource utilization

Md Razon Hossain [b], Md Whaiduzzaman [a,b,*], Alistair Barros [a], Shelia Rahman Tuly [b], Md. Julkar Nayeen Mahi [b], Shanto Roy [b], Colin Fidge [c], Rajkumar Buyya [d]

[a] School of Information Systems, Queensland University of Technology, Brisbane, Australia
[b] Institute of Information Technology, Jahanirnagar University, Dhaka, Bangladesh
[c] School of Computer Science, Queensland University of Technology, Brisbane, Australia
[d] Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Keywords:
Resource utilization
Task scheduler
Cloud computing
Fog computing

ABSTRACT

Fog computing is one of the most important emerging paradigms in recent technological development. It alleviates several limitations of cloud computing by bringing computation, communication, storage, and real-time services near to the end-users. However, with the rapid development of automation in smart cities, the number of task executions by fog nodes are increasing, requiring additional fog nodes. In this paper, we present a Scheduling-based Dynamic Fog Computing (SDFC) Framework to augment the utilization of existing resources rather than adding further fog resources. It includes an additional layer, Master Fog (MF), between the cloud and general-purpose fogs, which are addressed here as Citizen Fog (CF). The MF is responsible for deciding task execution in CFs and the cloud. We use the *Comparative Attributes Algorithm* (CAA) to schedule tasks based on their priority and a *Linear Attribute Summarized Algorithm* (LASA) to select the most available CF with the highest computational ability. Our empirical results validate our SDFC framework and show the dependency on the cloud reduces by 15%–20% and overall execution time decreases by 45%–50%.

1. Introduction

Automation and construction of smart cities have become essential components of modern civilization. Everyday, millions of new Internet of Things (IoT) devices are deployed, from personal use to industrial production [1–3]. According to Statista [4], the number of IoT-connected devices in previous and future years is shown in Fig. 1.

This rapid increase in IoT devices creates large amounts of data. A large portion of this data is *big data* that requires highly powerful computing units for processing. Additionally, many devices require real-time services and higher accuracy [5] in object identification and decision-making, causing a direct impact on the internet and data centers [6], including the cloud. The cloud supplies availability of data, computing units, software, and infrastructure with security, reliability, and flexibility. However, cloud computing has several limitations, including network congestion, request–response latency in real-time services [7], etc. Security for a distant computing unit operating over the internet may cause security vulnerabilities [8], and man-in-the-middle [9] attacks can cause serious security breaches. As a solution, Cisco coined the term *fog computing* and offered a definition for fog computing that any device with computing, storage, and network connectivity, can be deployed anywhere with a network connection and extends

* Corresponding author at: School of Information Systems, Queensland University of Technology, Brisbane, Australia.
E-mail address: md.whaiduzzaman@qut.edu.au (M. Whaiduzzaman).

https://doi.org/10.1016/j.simpat.2021.102336
Received 24 July 2020; Received in revised form 22 March 2021; Accepted 16 April 2021
Available online 27 April 2021
1569-190X/

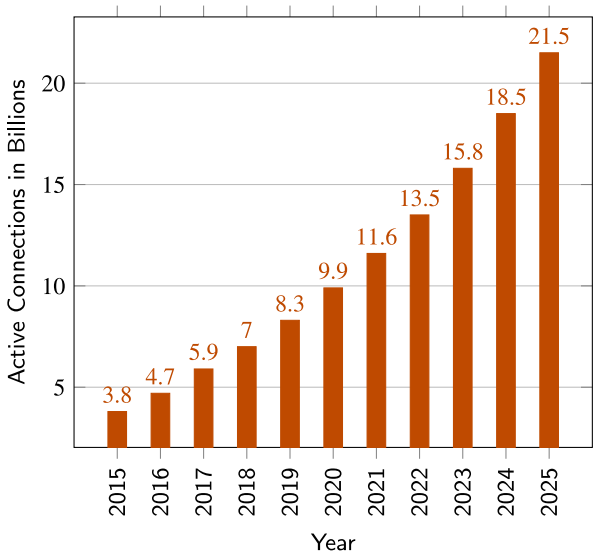


Fig. 1. Increasing number of IoT devices.

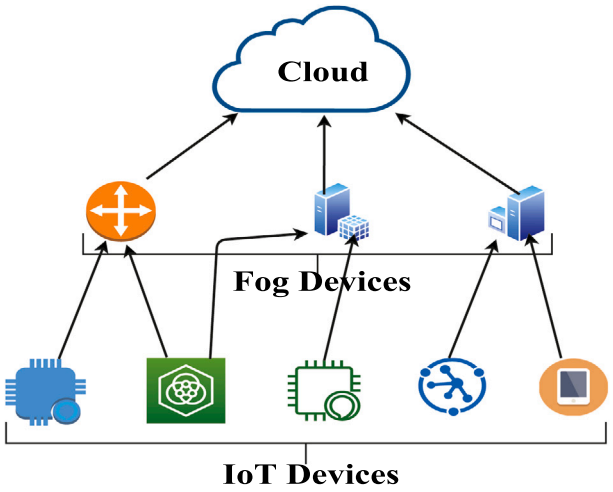


Fig. 2. Fog computing generic view.

the cloud near to IoT devices is a fog [10]. This approach offers advantages for services in several domains, including the smart grid, Wireless Sensor Networks (WSNs) [11], the IoT, and Software Defined Networks (SDNs) [12], Service Delivery Framework (SDF) in Software Oriented Architecture (SOA) [13], service synchronization [14,15], traffic systems and so on. Fog computing receives a request from end devices, such as the IoT, and processes it to supply a real-time response back to the requester [16,17] and also decides whether to send the computation to the cloud or not. Fog computing supplies data, computation, storage, and application services to the end-users, similar to the cloud [3]. Bittencourt et al. [18] provided a generic overview of fog computing which is shown in Fig. 2.

Although the concept of fog computing is promising, with time, the increase in data and required resources to process it grows too frequently for fog computing to be effective. According to the International Data Group (IDC), 30% of generated data will be real-time data by 2025 [19]. Hence, it shows the urgency of an alternative approach.

For instance, in a traffic control system [20], fog devices are integrated into each lamp post of the road. In the day time, these roads become highly dense with vehicles and smart devices. As a result, the amount of generated data is very high and requires executing different operations, which results in a data overflow of task within the queuing of the corresponding fogs [21–23]. A general solution to meet this required computation request is to increase the number of fog nodes or add additional processors to the existing fog nodes. For this increasing number of fog nodes, the conjoint network becomes complicated and more complex while demanding an excessive degree of maintenance. An increase in electronic equipment has a direct impact on environmental

and natural entities [24]. However, if we observe closely, we can find some idle or available fog nodes on the opposite side of the road, nearby homes, or institutes, etc. that can execute some tasks for other fogs. In the daytime, fogs at the side of the road, at the side of the nearby offices, industries, schools, etc. remain busier. On the other hand, home devices can be completely available for additional task processing at this time. However, at night the situation is reversed. Therefore, in different circumstances and times, some fogs are having a high overhead in executing a huge number of tasks whereas few fogs are available or idle. Consequently, a better solution is the proper utilization of the available resources.

Concerning the problem stated above, this research aims to provide a convenient and efficient framework that increases the reusability of existing resources. Our framework is a colonial framework where each colony consists of several existing general-purpose fog nodes named *citizen fog*. Moreover, each colony has a special fog node, called *master fog*, which maintains the corresponding citizen fogs. The range and the number of citizen fogs under a master fog are determined based on the density of data generated from the end devices, and a citizen fog can use the computational power and storage of another citizen fog via the master fog. The master fog node is responsible for load balancing and intercommunication among the citizen fogs by taking each request from a citizen fog and assigning the task to the most eligible citizen fog available. After task execution, the master fog returns the result or response to the requester citizen fog. The master fog keeps track of the computing status of the citizen fogs in real-time and schedules the tasks based on their priority. To accomplish these goals, the process follows our new Comparative Attributes Algorithm (CAA) to schedule tasks based on their priority and uses the Linear Attribute Summarized Algorithm (LASA) to sort the available CFs according to their computing abilities. Besides this, it is also responsible for offloading data to the cloud and maintaining communication between the citizen fogs and the cloud.

Contribution. In this paper, we depict the scattered resource distribution which causes inefficiency in handling the increase of data from IoT devices and describe our new SDFC framework that augments the reusability of existing resources. The major contributions of this paper are summarized as follows:

- We reconstruct the traditional fog-cloud framework into a three-layer framework *SDFC*, introducing a master fog that communicates with the cloud and CFs to share available resources and computing ability for executing tasks.
- We address the problem of scattered resources and design a *Comparative Attribute Algorithm* (CAA) for the master fog to maintain and sort the task queue based on priority and a *Linear Attribute Summarized Algorithm* (LASA) to sort out the appropriate fog for a specific task.
- We simulate our SDFC framework, compare it with a traditional three-layer fog computing framework, and validate the performance of our SDFC framework. It shows that our framework performs better than the alternatives in augmenting resource utilization.

Outline. The rest of the paper is organized as follows: Section 2 presents a brief overview of the literature and related work. We examine several papers and research studies to identify the current obstacles, solutions, and their limitations. Section 3 illustrates our new task scheduling fog computing framework, its layers and components, and their interactions and task scheduling parameters, and based on the parameters, we describe the process and scheduling algorithms and how they are used in the framework. Section 4 evaluates the simulated results of the framework and presents comparisons with existing works in different situations followed by our concluding remarks on our work by discussing major findings, and future implications.

2. Background and rationale

We have studied and analyzed different cloud-fog computing frameworks and scheduling algorithms. This section contains the review of these frameworks and algorithms along with the motive and necessity of our SDFC framework.

2.1. Cloud-fog computing frameworks

Bonnet, et al. proposed a two-layer framework installing a management layer in the fog and a data semantic knowledge and analysis layer on the cloud to store, analyze, and learn from the managed fog instances (FIs) data [25]. The main idea is that once a solution to a problem is processed, the fog layer manages it and supplies it to the further requests from devices. This approach integrates the Machine to Machine Management (M3) framework [26] at the Road Side Units (RSU) and the Machine to Machine (M2M) approach [27] to analyze the data and perform sensor-based tasks to offer the same solution to a new request that has already been computed, thus increasing real-time responses. However, the framework is completely dependent on the cloud. For the heterogeneous tasks, the *foggy* framework [28] proposed by Yigitoglu et al. is more effective. The proposed task scheduling mechanism optimizes the resource usage and minimizes the latency by considering application requirements from its latency, sensitivity, resource constraints, mobility support, dynamic workload, and privacy [29] as well as resource capabilities, costs, and mobility. Santoro et al. offered a framework on the workload of orchestration using the foggy framework [30]. Bruin et al. also proposed a coordinate-based foggy cloud and cloudy fog framework [31]. However, in this framework, each fogs are independently and directly connected with the cloud and there is no methodology to share resources. Therefore, fogs are completely dependent on the cloud, if there is any idle fog nearby, other busy fogs cannot take advantage of this. A Green and Timeliness-oriented Fog Computing Model called Spatio Fog [32] is provided by Das et al. which reduces power consumption and delay in geospatial query resolution system. To meet the challenges of fast-growing users traffic, Habibi et al. [33] proposed a four-layer architecture containing end device, fogs, core network, and cloud. It integrates and extends the ETSI NFV [34] and SDN [35]

architecture. Another three-layer hybrid approach is provided by Velasquez et al. [36]. The authors use an orchestrator to manage the communication, resource, service discovery to lookup service availability in the nearest location. Different orchestrator components are in a different layer and cloud contains a single instance for global orchestration. Brito et al. [37] proposed another orchestration based architecture, named Service Orchestration Architecture for Fog-enabled Infrastructures (SOAFI) to orchestrate fogs. It has two main components, a Fog Orchestrator and Fog Orchestration Agent. The Fog Orchestrator converts the fog nodes into logical infrastructure that makes an inventory of available resources in the fog domain and allocate them. Each cloudlet contains a Fog Orchestration Agent and provides the management interface to the Fog Orchestrator.

2.2. Scheduling based fog computing frameworks

Pham and Huh presented a task-scheduling algorithm based on trade-offs and proposed a three-layer framework [38] in which a smart gateway communicates between the fogs and cloud. In this task scheduling cloud-fog computing system, a fog provider can use the collaboration between its fog nodes and rent cloud nodes to execute requests while all the fogs are busy. Choudhari et al. proposed a priority-based scheduling algorithm in [39] based on the Efficient Resource Allocation (ERA) [40] algorithm by applying prioritized scheduling. The priority queue depends on whether a particular fog can complete the task within the allowed delay, estimated service time, and threshold time. The authors asserted that their algorithm significantly reduces the execution cost. However, the priority scheduling is occurring in each fog which causes delays in real-time response. Verma et al. proposed a Real-Time Efficient Scheduling (RTES) algorithm for load balancing in the fog computing environment [41]. The authors asserted that their scheduling algorithm ensures that all processors in the systems and every node in the network share an equal amount of the workload requested by the client. The framework uses the load balancing algorithm in the fog computing ecosystem to improve real-time services. However, it is not concerned about the heterogeneous resource capability of fog devices, and in the case of a large amount of data, it continues to search for fog devices in the other regions, which delays the real-time services. Chu et al. focused on the visual fog scheduling problem of assigning the visual computing tasks to various devices for optimal network utilization [42]. The work of these authors proved that this problem is NP-complete, which is different from the usual scheduling, and they formulated a practical, efficient solution. The main idea was to divide the tasks and devices in parallel and map the tasks to the devices in parallel as well. However, problems arise when the slices cannot work independently and there is a dependency between tasks. Therefore, although the task holds the resource, it cannot complete the execution.

2.3. Generalized scheduling algorithms

There are various scheduling algorithms such as resource management [43], Minimum Completion Time (MCT), and Minimum Execution Time (MET) algorithms [44], and MIN-MIN and MAX-MIN algorithms [45]. MCT does not maintain any scheduling or sequence for the processors or tasks. Instead, it assigns tasks to the nearest processors, without confirming the execution of the task. MET assigns tasks to the most powerful processor for fast execution. However, load balancing is not stated here. HealthEdge [46] collects human health data to set various processing priorities to schedule tasks and determine which tasks should be executed locally and which should be executed in the cloud. Mijumbi et al. proposed a Greedy Best Availability (GBA) [47] process to define an ideal task scheduling policy, and based on work completion time, they attempted to reduce the queuing time. Al-khafajiy et al. provided a fog upload algorithm in their Fog Computing Framework for IoT application [48] which decides which task to offload and where to offload based on the task congestion and fog availability. The execution of the algorithm occurs in every fog which causes computation overhead. Moreover, Tychalas provided a Bag-of-Tasks [49] scheduling algorithm that uses all the available resources to decrease cost. However, the algorithm sends a set of tasks to each resource and task to stay in a queue before executing, and thus increases the response time.

In the literature review, we have discussed the advantages and disadvantages of several frameworks and algorithms. However, there is still a lack of proper resource utilization and delays for high-priority task execution. For example, the RTES lacks the balance of proper resource utilization and real-time services. Therefore, we develop the SDFC framework which provides the ability to use the resources of adjacent CFs where sharing and task scheduling are handled by a new layer, titled Master Fog. The Master Fog is similar to any other citizen fog regarding its resources with a list of selective responsibilities to track the CF resource status and sort CFs using the LASA algorithm, scheduling and assigning tasks based on their priority by using the CAA algorithm and communicating with CFs and cloud. It stays as close to the end device as a citizen fog stays. Therefore, the computation overhead is minimal. Moreover, it utilizes the resources of adjacent CFs instead of renting cloud resources all the time. Therefore, the cost is minimal and execution time is faster, and hence reduces delays in real-time responses.

3. System model and methodology

In this section, we describe our SDFC framework and its components, and we present the mathematical model for scheduling tasks.

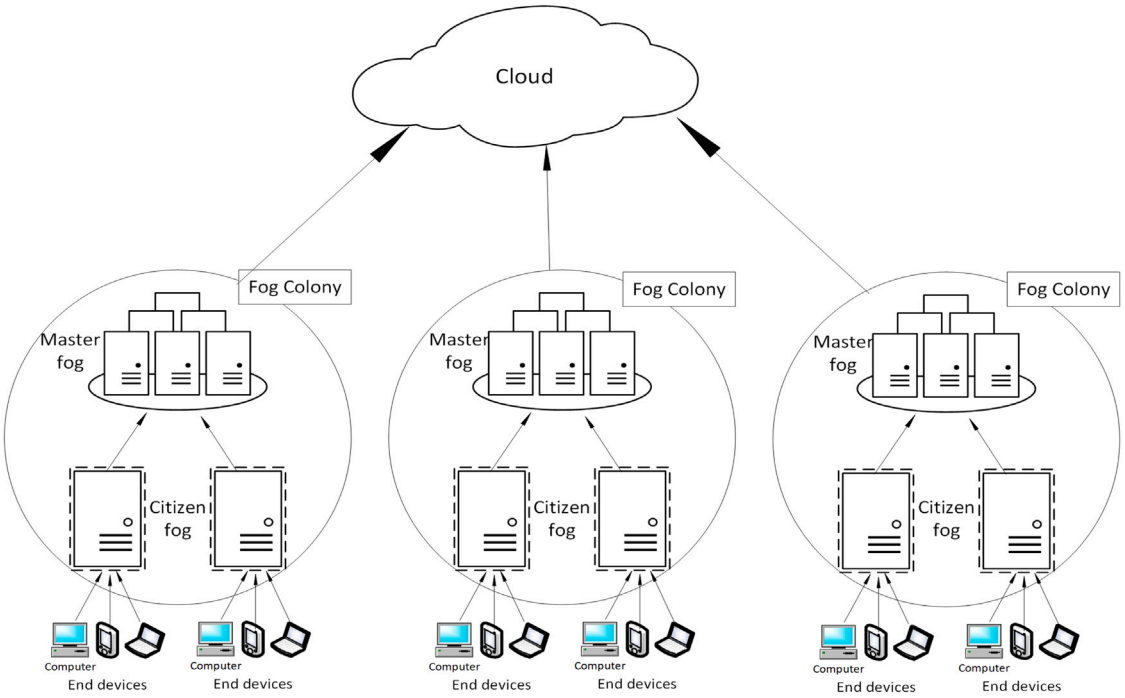


Fig. 3. Overview of SDFC framework with citizen fog, master fog, and fog colony concept.

3.1. Task scheduling fog computing framework

Fog computing is a recent paradigm, and therefore, many challenges and opportunities remain to cope with the increasing requirement for resources and computation power. Researchers are attempting to overcome these challenges and looking for optimized utilization of the resources to ensure better service and computing for IoT and edge devices (e.g., mobiles, laptops). We added a new layer of a particular fog device between the general-purpose fog devices and cloud. This three-layer framework places additional attention to processing and maintaining the complexity carried by the fog and cloud. Here, the workload is mostly considered for the fog by classifying fog nodes according to their *primary ability*, as stated by Yigitoglu et al. [28], and a *context-aware queuing* system for tasks coming into the master fog, as stated by Mostafa and Mohammad [50]. We have retained the end devices as isolated and encapsulated from other fog layers to create a decoupled relationship between them.

These three layers (except for the user layer) consist of the fogs and cloud. The relationship between the end-user and our first layer (general-purpose fog nodes) is retained as before. Our research results offer a better management process between the fog and cloud that can reliably supply proper resource utilization, resulting in *faster responses* and *fewer request failures* for the end-user, although they work with fog in the traditional manner by remaining unaware of the changes in the framework.

The first layer of the framework contains the general-purpose fogs, named as a citizen fog, which are the nearest nodes to the end-user and usually control the communication between the cloud and the end-user through the master fog, when necessary. This layer can manage subsequent computing on behalf of the cloud and supply real-time responses to the user but due to the inconsistency of the *computing request density* over places and time, a dynamic task allocation controller for the citizen fogs must be introduced.

The second layer can dynamically handle the request traffic that the citizen fog cannot execute. This layer is marked as the master fog. The Master Fog (MF) creates a colony consisting of a few citizen fogs, and it aims to receive a request from one fog and allocate the task to another fog on behalf of the requesting fog. After task completion, this layer receives the response from the worker fog and redirects the response to the requesting fog.

The Cloud is the third layer of our SDFC framework. It communicates with the Master fog. If there is no citizen fog available to execute a task, the Master fog sends the task and reference to the requesting Citizen fog to the cloud and after completion of execution, the cloud sends the response to the referenced citizen fog directly.

The SDFC system model is shown in Fig. 3.

3.2. Component description

Fig. 4 shows the components of our solution and their intercommunication in depth.

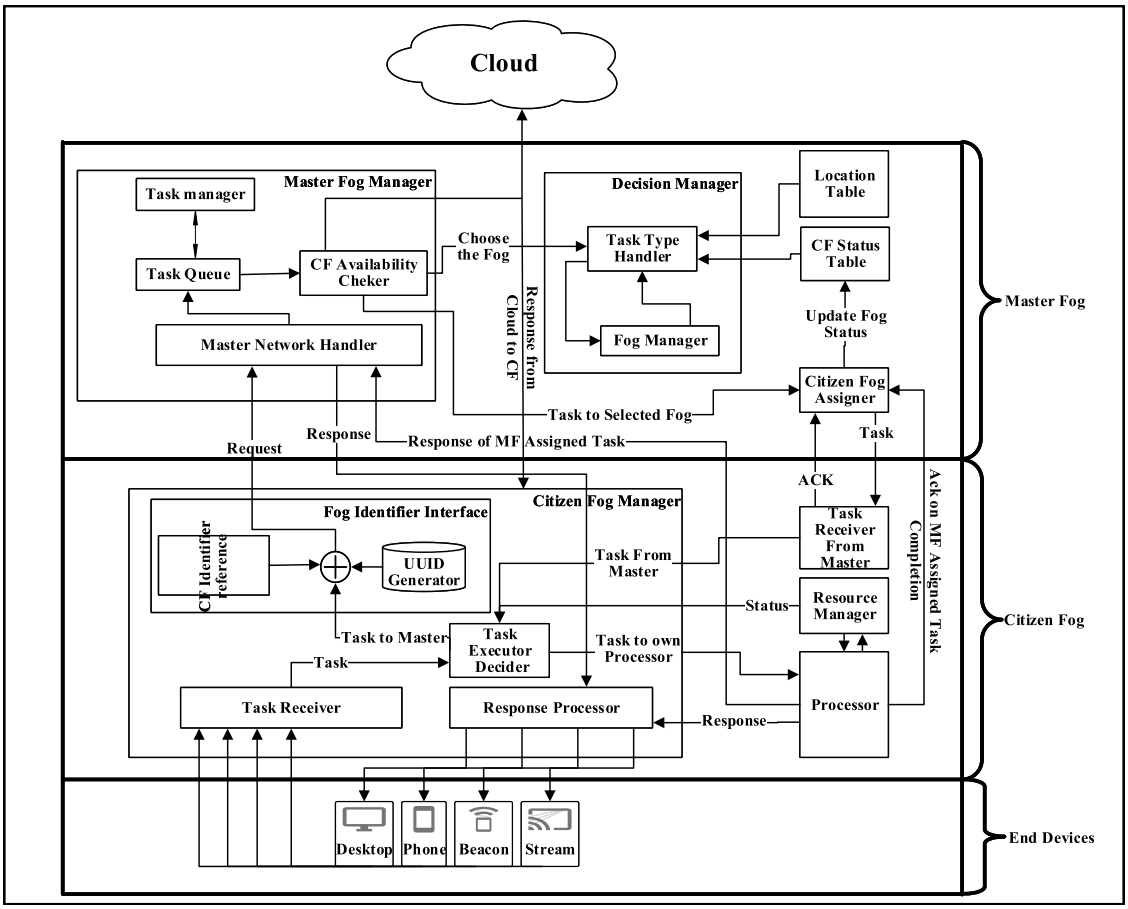


Fig. 4. Our task scheduling fog computing framework.

3.2.1. Citizen Fog (CF)

The first layer of our SDFC framework consists of Citizen Fogs. These fogs are the closest devices to the end-users. Smart devices, laptops, and IoT devices, etc. are connected with these fogsfogs in the same way they usually connect with any other general-purpose fog. Receiving tasks from end-users and the master fog, executing them, and sending the executed result to the corresponding CF are its responsibilities. Moreover, if it cannot execute a task, it asks the master fog to execute the task. A CF consists of a Citizen Fog Manager (CFM), the controlling unit of the fog. Besides this, there are processors as a computing unit, Task Receivers from the Master Fog, and Resource Managers to manage the resources of the CF.

Citizen Fog Manager (CFM). This is the central controlling unit of the citizen fog. It consists of a Task Receiver (TR), a Fog Identifier Interface (FII), a Task Executor Decider (TED), and a Response Processor (RP). It manages and routes the tasks using Algorithm 1. The TR receives a task from an edge device and adds it to the task queue of the TED. For each task from the edge-device, the TED gets the resource status from the Resource Manager and checks the ability to execute the task. If the execution is possible, it sends the task to the processor. Otherwise, it sends the task to the FII to send to the master fog. Moreover, it allows the task from the master fog to execute in the processor immediately because the master fog sends the task to a citizen fog only if the citizen fog has sufficient resources to execute the task. After executing a task requested from the Master Fog, the processor sends the executed result to the Master Network Handler (MNH), and MNH sends the result to the Response Processor (RP) of the corresponding CF. RP receives the response from the MNH for the task the CF requested to MF and then, it distributes the responses to the corresponding end-device.

Fog Identifier Interface (FII). When a citizen fog lacks the resources to execute a task, the FII attaches a unique identifier and fog reference to this task and sends it to the master fog. The FII also receives the response from both the master fog and the cloud. Upon receiving the response, it uses the unique identifier to identify the task the response is mapped to.

3.2.2. Master Fog (MF)

The Master fog (MF) is the core component of our SDFC framework. It maintains communication with other citizen fogs and the cloud. The master fog creates a colony, consisting of multiple CFs. The number of CFs depends on the density of the computation

Algorithm 1 Citizen Fog Task Management

```
1:  taskQueue = Queue of tasks from end-devices
2:  CF = Citizen Fog
3:  MF = Master Fog
4:  procedure CitizenFogManager(task)
5:      while taskQueue ≠ 0 do
6:          task ← taskQueue.pop()
7:          if taskIsFromEdgeDevice() then
8:              if isCFCanExecute(task) then
9:                  computeTheTaskInProcessor(task)
10:                 sendResourceStatusToMF()
11:             else
12:                 askFIItoAddIdentifierTo(task)
13:                 sendTaskToMF(task)
14:         else
15:             computeTheTaskInProcessor(task)
16:             sendResourceStatusToMF()
```

requested by the end-devices. For instance, computation requests from end-devices vary in roads, residential areas, economic areas, industrial areas, and rural areas. All the CFs must be within the MF's coverage area.

Receiving and scheduling tasks from CFs, managing and keeping track of the resource status of the CFs, assigning a task to the most eligible CF or the cloud, and sending execution results as a response to the corresponding CF are its primary responsibilities. The Master Fog Manager (MFM) and Decision Manager (DM) are its major components. Furthermore, it consists of the Location Table, CF Status Table, and Citizen Fog Assigner (CFA). The CFA assigns each task to the most eligible CF and receives the acknowledgment from the assigned CF with the resources.

Master Network Handler (MNH). The Master Network Handler (MNH) is the communication module of the Master Fog. It accepts the task request from the FII of a CF and adds the task to the task queue of the Master Fog Manager (MFM). Furthermore, it receives the execution response of an assigned task from a CF and sends the response to the corresponding CF. To handle these incoming and outgoing operations, it follows the Message Queuing Telemetry Transport (MQTT) protocol [51], which results in a low latency response to the requesting CF.

Master Fog Manager (MFM). The Master Fog Manager (MFM) is the management component of the Master Fog. This module consists of the MNH, Task Queue (TQ), Task Manager (TM), and CF Availability Checker. The MFM checks for the availability of an eligible CF for each task in the Task Queue and assigns the task to the CF. If no CF is available, it sends the task to the cloud. Furthermore, it is responsible for managing CFs and assigning tasks. It receives resource statuses from the CFs and updates the CF Status Table. The MFM follows Algorithm 2 to accomplish its responsibilities. For each task in the TQ, it asks the Decision Manager (DM) to provide a CF. The DM performs various calculations to define the most eligible CF and returns to the MFM. It assigns the task to the CF and updates the CF status table with resource information in acknowledgment of the assignee CF. If the DM returns no suitable CF for a specific task, it sends the task to the cloud. Moreover, the MFM pushes the task to the waiting queue if neither a CF is available nor the cloud. Besides this, for each completed task, the MFM updates the CF status table. Therefore the latency of assigning a task to the eligible CF decreases and improves the performance of the latency aware real-time services.

Task Queue (TQ). The Task Queue (TQ) has three sorts of priority queues, a queue for the incoming tasks, a queue for the waiting tasks, and another queue for the completed task. Whenever the TQ receives a task from the MNH, it pushes the task to the incoming task queue and the Task Manager (TM) sorts the queue according to our Comparative attributes algorithm (CAA) and schedules the tasks accordingly. The process of the TM and the CAA algorithm is discussed in the next section. If the MFM cannot find any suitable CF for a task, the TQ pushes the task into the waiting queue. Moreover, the completed task is pushed into the completed task queue.

Location Table (LT). This table contains the current locations of the CFs to determine their position. The location of CFs of an MF is listed as latitude and longitude. MFM uses Dijkstra's algorithm to find the shortest distance between the MF and correspondent CF and ensures that the CF is inside the predefined region of MF. This region for a particular MF is defined at the time of MF and CF placement.

Fog Status Table (FST). The table contains the resource and computing status of each CF under the corresponding MF. It consists of the available amount of RAM, the number of Processing Elements (PEs), Bandwidth (BW), and processing speed, expressed as Millions of Instruction Per Second (MIPS). The decision Manager calculates these resources and defines an efficiency point to sort the CFs. Whenever a CF executes a task or completes its execution, in other words, if there is any change in the resources of the CF, it notifies the MF and sends the resource status to update the CF status table.

Algorithm 2 Master Fog Task Management

```
incomingTaskQueue = Queue of tasks from CF.
ackCF = Acknowledgment from CF with resource status.
waitingQueue = Waiting queue for tasks that could not be assigned to any CF.
computedTaskQueue = Queue for the tasks that has completed execution.
result = Execution result of a task.
cfResource = Resource of a particular citizen fog.
procedure MasterFogManager( )
    while incomingTaskQueue ≠ 0 do
        task ← incomingTaskQueue.pop()
        if isComputableCFAvailable(task) then
            ackCF ← sendTaskToCF(task)
            updateCFStatusTable(ackCF.CFResource())
        else if isPossibleToSendToCloud(task) then
            sendTaskToCloud(task)
        else
            waitingQueue.push(task)
    while computedTaskQueue ≠ 0 do
        computedTask ← computedTaskQueue.pop()
        result ← computedTask.result()
        sendComputedResultToRequestedCF(result)
        cfResource ← computedTask.CFResource()
        updateCFStatusTable(cfResource)
```

Decision Manager (DM). The Decision Manager (DM) is the CF management module of the Master Fog. It consists of a Task Type Handler (TTH) and a Fog Manager (MF). The TTH has different categories of CFs according to different types of tasks. The working process of TTH and FM is portrayed in Fig. 5.

Whenever the TTH receives a task from the Task Manager, it picks the CF with the highest computing and resource capability from the corresponding task type CF category. If a task with a new type arrives, the TTH sends the type to the Fog Manager (FM) and the FM provides the list of CFs that can execute the task. Thereafter, the TTH adds this list as a new category, fetches the data from the CF Status Table, and sorts them using our Linear Attribute Summarized Algorithm (LASA). Moreover, when a task is assigned to a CF, it uses the HBB method [52] to receive the status from the acknowledgment of the assignee CF and updates the corresponding CF category.

3.2.3. Cloud

The third layer of our SDFC framework is the Cloud. Instead of receiving tasks from general-purpose fogs or end-devices [53], it receives tasks from the Master Fog. The MF sends tasks to the cloud only if there is no CF available and as the task is assigned to the cloud, there is no need to keep track of resource consumption for the task. Subsequently executing the task, the cloud sends the execution's result directly to the corresponding CF.

The main focus of our framework is to execute the task as close to the end user as possible and ensure the proper utilization of available resources to provide a latency aware real-time service. To do that, we emphasize use of fog devices rather than the distant cloud and low computing end-devices. The MF is introduced to handle the heavy task of resource distribution and task management. A CF status table keeps the resource and computing status of each CF updated and the Decision Manager keeps the CFs prepared according to the task types. Therefore, if any CF fails to execute a task, the MF assigns the task immediately to another CF.

3.3. Task scheduling mathematical model

The execution time of a task depends on the resources and computational ability of a CF. A CF with higher resource and computation ability executes a task faster than others. Besides this, all the tasks are not equally important. If we consider a traffic system, identifying an ambulance and clearing forthcoming traffic for the ambulance is more important than maintaining overall traffic flow. Furthermore, executing a task with higher priority with a CF with higher resource and computation ability reduces latency in real-time services. The abbreviation and mathematical notation of the algorithms are accumulated in Table 1.

In our SDFC framework, the Task Manager is scheduling the tasks according to their priority and the Decision Manager is sorting the CFs according to their resource capacity and performance. The dependency attributes to determine the priority of tasks and attributes to sort the CFs are as follows:

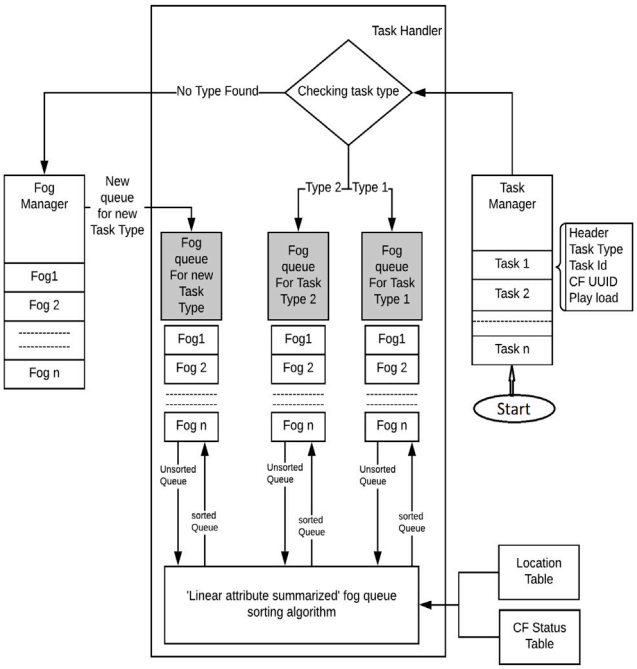


Fig. 5. Decision manager working process.

3.3.1. Capacity and performance

Fog capacity is the number of resources available for a specific task, such as MIPS, RAM, CPU, and the clock speed of the processor. The computational capability of a fog device depends on a specific set of resource constraints such as RAM, CPU, Bandwidth, and Processor Performance. To compare the performance of a particular CF, we have considered the *Execution Time* to execute a task with the base requirement. We have considered these attributes to sort the CFs according to their computing ability.

Execution Time. We use the *Performance Equation* [54] to calculate the estimated execution time. To compare the execution time to execute a specific task of each CF, we are calculating the ratio of the execution time for each CF with the Minimum Execution Time (MET) of a particular CF. We considered the unit task with 10^9 instructions. For a CF c, the execution time according to the Performance Equation is

$$T_c = IC \times CPI \times CT \tag{1}$$

where,

$$T_c = \text{Execution time for c}$$
$$CT = \frac{seconds}{clocks}$$
$$CPI = \frac{clocks}{instruction}$$
$$IC = \frac{instruction}{program}$$

This execution time T_c is different for different configuration. However, our aim is to find the device that can execute a number of tasks faster than other devices, independent of configurations. Therefore, we are considering the overall execution time only. If a device executes a certain number of tasks faster than other devices, gets a higher point.

Now, for n number of CFs,

$$MET = min \bigcup_{i=1}^{n} T_i \tag{2}$$

where, T_i = Execution time for ith CF.

Therefore, the Execution Time Ratio (ETR) for a CF c is

$$ETR_c = \frac{T_c}{MET} \tag{3}$$

Table 1
The abbreviations in the framework and notations used in the mathematical model and algorithms.

Symbol	Description
System and framework model	
CF	Citizen Fog
MF	Master Fog
CAA	Comparative Attributes Algorithm
LASA	Linear Attribute Summarized Algorithm
FII	Fog Identifier Interface
TED	Task Executor Decider
MNH	Master Network Handler
MFM	Master Fog Manager
TM	Task Manager
DM	Decision Manager
TQ	Task Queue
CST	CF Status Table
Algorithms	
c	Citizen Fog
T_c	Execution time for c
IC	Instruction Count
CPI	Clock Per Instruction
CT	Clock Time
MET	Minimum Execution Time
ETR_c	Execution Time Ratio of c
U, V, m	Task
W_u, W_v, W_m	Workload for u, v, and m
$n(m)$	Number of common subtasks in m
M_s	Set of subtasks of m
W_{u_s}	Net workload for W_u
MW	MinimumWorkload
WLR	Workload Ratio
NRR	Number of Request Ratio
MNR	Minimum Number of Requests
N_i	Number of requests of the ith task
D_u	Number of tasks depend on u
S	Set of depending tasks
D_{uv}	Task v depends on u
RR	RAM Ratio
PR	Processor Ratio
BR	Bandwidth Ratio

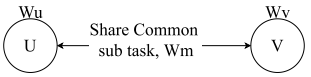


Fig. 6. Common workload.

3.3.2. Priority

Consider a social media system where all users must log in or sign up to go to the newsfeed page or profile page. Therefore the newsfeed module depends on the authentication module. Besides, the user does not login each time they access the newsfeed, they do it once but requests for newsfeed several times. Therefore to improve the browsing experience, the newsfeed page is required to load faster which implies being executed in a fog that has higher resource capability. In other words, all the tasks are not equally important. Consequently, we have considered the number of requests for each task, inter-dependency among the tasks, approximate time to execute a task, and workload or task-length of the task to prioritize the tasks and sort the tasks according to their priority.

Workload or task length. Consider two tasks u and v in Fig. 6. W_u and W_v are their workloads respectively. Both the task requires to execute a common sub task m with workload W_m. Therefore, if u executes m, v does not need to execute it, and vice-versa, consequently, reducing the workload of a task.

$$W_{u_s} = \begin{cases} W_u - \sum_{i=1}^{n(m)} W_{m_i} & \text{if } v \text{ executes } m \\ W_u & \text{if } u \text{ executes } m \end{cases} \tag{4}$$

where,

$m_i \in M_s$

M_s = Set of common sub tasks for the task u

$n(m)$ = Number of common subtasks of u and
W_{u_s} = Net workload for W_u

Likewise, for the ETR, to compare the workload of each task, we determine the Minimum Workload (MW) of a task and calculate the Workload Ratio (WLR) of the workload of each task against MW.

$$WM = min \bigcup_{i=1}^{n} W_i \tag{5}$$

Therefore,

$$WLR_u = \frac{W_{u_s}}{WM} \tag{6}$$

where,

W_i = Workload of ith task and
WLR_u = Workload Ratio for task u

Number of requests for each task. All the tasks are not equally requested by the end-devices. In a social media system the newsfeed module or a service system, a dashboard is frequently used. In contrast, a personal profile module or signup module is requested much less often than that. Moreover, the authentication module is necessary for each operation in the system. Likewise we did for the ETR and WLR, we calculate the Number of Requests Ratio (NRR) to compare the task requests. Therefore, the Minimum Number of Requests (MNR) for ith task,

$$MNR = min \bigcup_{i=1}^{n} N_i \tag{7}$$

where n is the number of tasks and N_i is the number of requests of the ith task. Therefore, the Number of Requests Ratio (NRR) for ith task,

$$NRR_i = \frac{N_i}{MNR} \tag{8}$$

where N_i = Number of request for ith task.

Dependencies. Different tasks are dependent on one another. In any secure system, a transaction depends on the security module. The user's financial data and transaction media must be secure and needs to be applied before the transaction can execute. To make way for an ambulance, the destination hospital must be known. Consequently, the task, on which other tasks depends, needs to execute faster and before the other task is executed. We calculate the priority of a particular task based on these dependencies using the arithmetic sum of each task that depends on this particular task. That is, for a task u, the dependency point D_u is

$$D_u = \sum_{i=1}^{n(S)} i \tag{9}$$

where, S = Set of tasks that depends on u and $i \in \mathbb{Z}$

3.4. Process and algorithm

Based on the attributes and challenges in the previous subsection, we construct two algorithms, CAA and $LASA$. The Task Manager and the Task Type Handler of the Master Fog execute these algorithms respectively. To execute LASA, the TTH requires the Fog Status Table to be updated always. The Master fog does this update task as well. It subscribes the resource module of each citizen fog and whenever any changes occur in the resource of a citizen fog, the master fog is notified. We follow the MQTT protocol for this subscriber–observer actions. These three are the major activities of the Master Fog.

3.4.1. Comparative attributes algorithm (CAA)

We have considered four attributes to define the Comparative Attribute Algorithm (CAA) 3: tasks Dependency on each-other, the Number of tasks dependent on a particular task, the NRR, and the WLR. Whenever a TQ receives a task, it sends the incoming task queue to the TM. The TM prioritize the queue based on these aforementioned attributes. To do so, the TM uses a Divide and Conquer process. Divide and conquer recursively divides the queue until there are two adjacent tasks. Then it compares these two tasks based on our defined attributes. In this comparison, we use our CAA algorithm.

First of all, it compares whether there is any dependency between the two tasks or not. If yes, the independent task is selected as the prioritized task. If there is no dependency, then the CAA compares the number of other tasks dependent on each of these tasks. The CAA prioritizes a task with higher dependency points than all another. Consider Fig. 7, task u depends on v. Therefore, $D_u = 0$ and $D_v = 1$. On the other hand, u and n are independent. However, both n and u depend on m, hence, $D_m = 2$. Consequently, in the case of u and v, v is the prioritized task and in the case of v and m, m is the prioritized task.

Afterward, if both the tasks have equal dependency points, the CAA compares the Number of Requests Ratio (NRR). The task with the higher NRR is the high priority task. Subsequently, if NRR is equal, the CAA compares the WLR of the adjacent tasks.

Algorithm 3 Comparative attributes algorithm (CAA) using divide and conquer

Step 1:
  Input: UT = list of unsorted tasks, $n = UT.size()$
  Output: A sorted UT
Step 2: Using CAA in $MergeSort$ until UT is sorted.
Step 3: Inside the comparison of Merge sort: u and v are two tasks to *compare*, P = prioritized task to *return* from CAA ▷ Merge sort uses divide and conquer and sorts a list by dividing it into a number of pair, compare between them and continues merging.
procedure CAA(u, v)
  if D_{uv} exists **then**
    $P \leftarrow u$
  else if D_{vu} exists **then**
    $P \leftarrow v$
  else
    Compute D_u and D_v from Eq. (9)
    if $D_u > D_v$ **then**
      $P \leftarrow u$
    else if $D_u < D_v$ **then**
      $P \leftarrow v$
    else
      Compute NRR_u and NRR_v from Eq. (8)
      if $NRR_u > NRR_v$ **then**
        $P \leftarrow u$
      else if $NRR_u < NRR_v$ **then**
        $P \leftarrow v$;
      else
        Compute WLR_u and WLR_v from Eq. (6)
        if $WLR_u > WLR_v$ **then**
          $P \leftarrow v$
        else
          $P \leftarrow u$
  return P
End

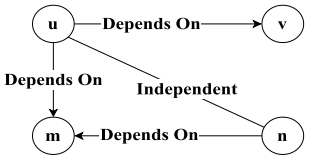


Fig. 7. Dependencies among tasks.

Complexity of CAA. For n tasks, the CAA calculates D, NRR, and WLR of a specific task while comparing. However, each of these operation calculates their ratio against other task. Consequently, the complexity of each operations is $\mathcal{O}(n)$. Besides this, to apply divide and conquer, we use *Merge Sort*. This algorithm divides our task queue into the size of the two tasks and compares them. The complexity to compare between the attributes of two tasks is $\mathcal{O}(1)$. The complexity of the merge sort is $\mathcal{O}(n \log n)$. Consequently,

$$\begin{aligned}\mathcal{O}(CAA) &= 3 * \mathcal{O}(n) + \mathcal{O}(1) + \mathcal{O}(n \log n) \\ &= \mathcal{O}(3n + 1 + n \log n) \\ &= \mathcal{O}(n \log n)\end{aligned}$$

3.4.2. Sorting citizen fog in master fog manager

The Task Type Handler (TTH) sorts the CFs using our *Linear attribute summarized algorithm ($LASA$)*. Primarily, all the CFs are stored in Fog Manager (FM) in $FCFS$ pattern. When a task arrives at the TTH, the FM returns the list of CFs that can execute the type of that task. Afterward, the TTH applies LASA to sort the CFs and assigns the task to the CF which is on top of the sorted queue. It follows the *HoneyHoney Bee Behavior inspired Load Balancing (HBB-LB)* method [52], that is when a CF is assigned with a task, it allocates resources to the task and returns an acknowledgment. This acknowledgment contains the resource status of the CF and the TTH updates the sorted queue of that specific task type category.

LASA considers four attributes of a CF, the number of available processors, the time to execute a task (ETR), available RAM, and available Bandwidth. Similar to ETR, we define the Processor Ratio (PR), RAM Ratio (RR), and Bandwidth Ratio (BR) by calculating the available overall resource against the minimum amount.

Therefore, for a particular task the Execution Time Ratio (ETR) for a CF c from Eq. (3)

$$ETR_c = \frac{T_c}{MET}$$

where MET = Minimum Execution Time among the CFs. It is calculated in Eq. (2). Likewise, we perform the following calculations. For RAM,

$$RR = \frac{RAM_c}{min \bigcup_{i=1}^{n} RAM_i} \tag{10}$$

For Processor,

$$PR = \frac{PR_c}{min \bigcup_{i=1}^{n} PR_i} \tag{11}$$

For Bandwidth,

$$BR = \frac{BW_c}{min \bigcup_{i=1}^{n} BW_i} \tag{12}$$

where n is the number of CFs and $i \in \mathbb{Z}$. The *LASA* algorithm is presented in Algorithm 4

Algorithm 4 Linear Attribute Summarized Algorithm

1: **Input:** UF = list of unsorted fog, $n = UT.size()$
2: **Procedure:**
3: **Step 1:** Using $LASA$ in $Mergesort$ until UF is sorted
4: **Step 2:** Inside the comparison of Merge sort: Fog_A and Fog_B are two CFs to *compare*, F = The task with higher resource to *return* from $LASA$ ▷ Merge sort uses divide and conquer and sorts a queue by dividing it into a number of pair, compare between them and continues merging.
5: **procedure** LASA(UF)
6: Compute ETR_a and ETR_b from Eq. (3)
7: Compute RR_a and RR_b from Eq. (10)
8: Compute PR_a and PR_b from Eq. (11)
9: Compute BR_a and BR_b from Eq. (12)
10: Compute $Point_A = \frac{1}{ETR_a} + PR_a + RR_a + BR_a$
11: Compute $Point_B = \frac{1}{ETR_b} + PR_b + RR_b + BR_b$
12: **if** $Point_A > Point_B$ **then**
13: $F \leftarrow Fog_A$
14: **else if** $Point_B > Point_A$ **then**
15: $F \leftarrow Fog_B$
16: **else**
17: **if** $ETR_b \geq ETR_a$ **then**
18: $F \leftarrow Fog_A$
19: **else**
20: $F \leftarrow Fog_B$
21: return F
22: **End**

The algorithm uses *divide and conquers* which divides the CF list until there is two CF. Afterward, it compares those two CF and merges them by backtracking until the whole list is sorted. To compare, it calculates the ETR, RR, PR, and BR for each of the CF. In calculating the resources such as RAM, Bandwidth, and the Number of Processors, the more the better. However, in case of Execution time, the less the better. The Execution Time Ratio (ETR) shows how many times a CF requires to execute a specific task than the CF which requires the lowest execution time among all CFs in the list. Therefore, the *inverse of ETR* shows how much *less time* the CF with the lowest execution time requires. In the LASA we are adding the *inverse of ETR* with resources to define the points and sort in descending order. The CF with higher resources has higher RAM than others, better performance, therefore faster execution time than others, higher Bandwidth availability, and higher Processor performance than others. Consequently, the LASA considers these ETR, RR, PR, and BR values as *points* and sum them to calculate a point for a specific CF and return the CF with the highest point. However, if these points are equal, LASA compares the execution time and prioritize the CF with less execution time.

Table 2
System descriptions for simulation environments.

System properties	Specification
Operating System	Windows 10
Processor	i3-4100M, CPU 2.50 GHz
Storage	RAM 16 GB, SSD 256 GB
Number of Cores	4
System Architecture	64 bit, x64
Simulator	iFogSim
Programming Language	Java
IDE	Intellije IDEA 2019.2.4

Complexity. For n CFs, the $LASA$ calculates ETR, RR, PR, and BR values of a specific CF while comparing them. However, each of these operations calculates their ratio against other CFs. Consequently, the complexity of each operation is $\mathcal{O}(n)$. Besides this, to apply divide and conquer, we use the *Merge Sort*. This algorithm divides our CF list into the size of the two CFs and compares them. The complexity to calculate and compare the points between the CFs is $\mathcal{O}(1)$. The complexity of the merge sort is $\mathcal{O}(n \log n)$. Consequently,

$$\begin{aligned}\mathcal{O}(LASA) &= 4 * \mathcal{O}(n) + \mathcal{O}(1) + \mathcal{O}(n \log n) \\ &= \mathcal{O}(4n + 1 + n \log n) \\ &= \mathcal{O}(n \log n)\end{aligned}$$

In this section, we discussed our *SDFC* framework, its layers, and different attributes. Furthermore, based on the attributes, we discussed our priority-based task sorting algorithm *CAA* to schedule the tasks and resource-based CF sorting algorithm *LASA* to allocate the highest amount of resources to the task with the highest priority. The Master Fog does the heavy tasks to keep the CF list up to date continuously with the most qualified CF on top so that, whenever a high priority task arrives, it can be assigned and execute immediately. In the next section, we discuss the simulation and data gathering of our task-scheduling fog computing framework, *SDFC*.

4. Performance evaluation

In this section, we analyze and discuss the simulation results of our SDFC framework and its CAA and LASA algorithms and compare them with an implemented three-layer cloud-fog computing framework in iFogSim [55].

4.1. Simulation platform

Our SDFC framework contains three layers, Citizen Fog (CF), Master Fog (MF), and Cloud. The framework considers each CF's available resources and schedules tasks to maintain the proper utilization of the available resources. To simulate our scheduling based optimized resource allocation framework, we use the *iFogSim* simulator [55]. Moreover, to compare and evaluate our framework's performance, we consider a three-layer cloud-fog computing framework, simulated in the iFogSim simulator. The layers are end devices, gateway devices, and the cloud. The end device requests the gateway device for resources, and if the gateway device is not capable of providing enough resources, it redirects the task to the cloud. It is assigning tasks without considering any scheduling or priority and considered as First Come First Serve (FCFS) fog computing framework. However, similar to our SDFC framework, it also considers the resources of fog devices, though, unlike our SDFC framework, it does not follow any scheduling algorithm. Moreover, we compare the execution time with the RTES framework.

The *iFogSim* simulator has a Sensor class that emits tasks called Tuple, *FogDevices* process the task, and Actuator receive the processed task. We use the Object-Oriented programming feature, *Inheritance* to use the *FogDevice* class as CF, MF, and Cloud. However, the execution time calculated in Eq. (1) is different for different device configuration. Therefore, in the simulation, we consider same software environment and same citizen fog devices. In addition, iFogSim also generates similar tasks. The system specification is shown in Table 2. Moreover, Table 3 shows the CF and MF specification and, Table 4 shows the task required resource specification. The *PowerModelLinear* class of the iFogSim simulator is used to calculate the energy consumed by a node [56]. We consider the cloud with infinite resources. The simulation result depends on the system specification; therefore, we execute the simulation ten times with two minutes and consider the average of each attribute in the simulation's output. To differentiate the resources among CFs, we generate and allocate random values of RAM, MIPS, BW, and PE to a CF.

4.2. Result and analysis

The SDFC framework schedules the tasks according to their priority and sorts the CFs according to their resource availability. Therefore, whenever MF receives a task, it assigns the most eligible CF to the task immediately. We have analyzed 500 tasks in the simulation. To describe the impact of scheduling these tasks, we consider 4 random tasks stated in Table 5. Task 2 depends on Task 4, Task 3 depends on Task 2, and Task 4 depends on Task 1. Consequently, if we execute the tasks in FCFS fashion as displayed in

Table 3
Citizen fog and master fog resource and system specification.

Attribute	Specification
Number of CF under a MF	5
MIPS	1024 * Random(8, 12)
RAM	512 * Random(1, 3)
UpBW	10000
DownBW	270
Busy Power	87.59
Idle Power	82.44
Number Of PE	Random(1, 3)
Storage	1000000
Bandwidth	1024 * Random(8, 12)
Architecture	x86
OS	Linux
VMM	xen

Table 4
Required resource specification of task.

Attribute	Specification
Number of task	500
RAM	32
MIPS	1024
TaskLength	1024
Bandwidth	128

Table 5
Tasks in CAA before scheduling.

Task ID	Dependent on	Workload	Number of requests
Task 1	None	1000	3
Task 2	Task 4	1700	5
Task 3	Task 2	500	4
Task 4	Task 1	1250	3

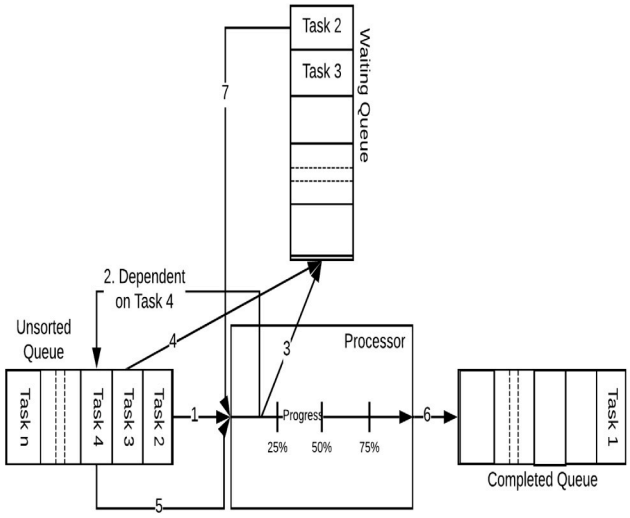


Fig. 8. Tasks executing in FCFS, creating waiting queue and delaying response time.

Fig. 8, Task 2 and Task 3 will be dismissed while executing and send to the waiting queue. However, after scheduling using CAA, the tasks are ordered and executed so that no task is obligated to send to the waiting queue, as portrayed in Fig. 9, which decreases the execution time for the task.

In the three-layer FCFS Fog Computing Framework in iFogSim, when a CF receives a task from the end device, it attempts to execute the Task. If the CF is unable to execute the Task, it transfers the Task to the gateway device. The gateway device tries to execute the Task as well, and when it cannot execute the Task, it forwards the Task to the cloud. Consequently, to execute a task, it requires waiting in CF, gateway, and, at last, in the cloud's queue. Fig. 10 shows the resource requirement and availability for

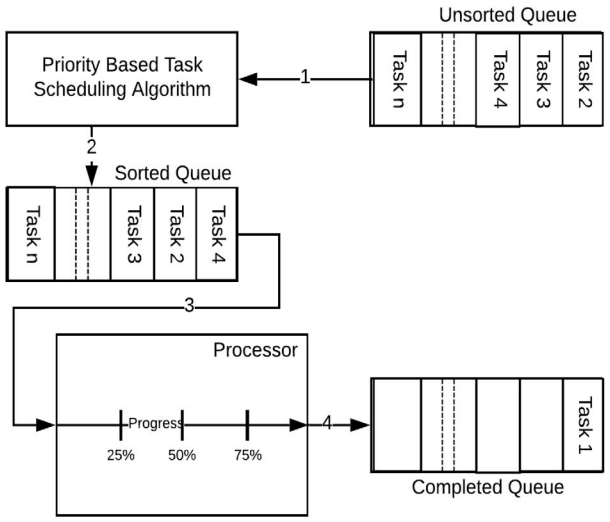


Fig. 9. Tasks executing in SDFC, no waiting queue and decreases response time.

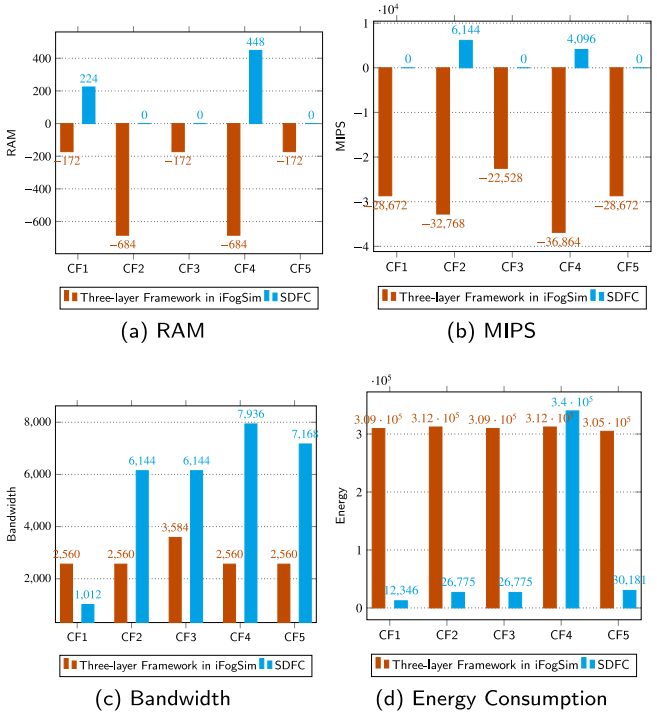
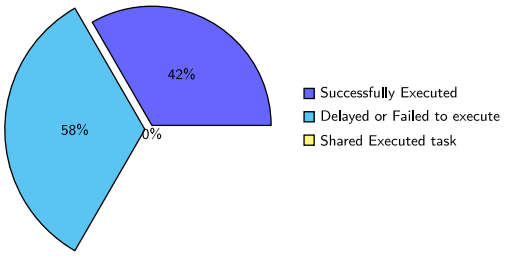
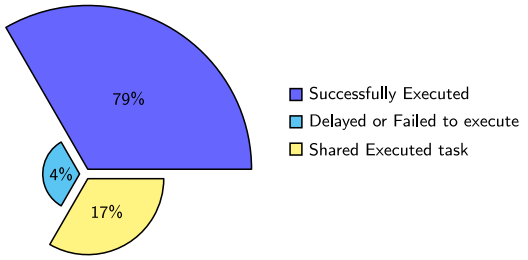


Fig. 10. Required and available resource comparison between SDFC and the three-layer Fog Computing Framework in iFogSim.

executing all the tasks in CF and gateway devices. The negative values signify the shortage of resources to execute all the tasks in the Fog Computing Framework in iFogSim. Consequently, the dependency on the cloud increase, and this causes delays in real-time services. Conversely, our SDFC framework uses the LASA algorithm to ensure the proper utilization of all the available resources of other CFs. If a CF is unable to execute a task, it immediately sends the Task to the MF, and MF assigns it to the most eligible CF from FST. This process continues until all the resources are used. Fig. 10 shows that many resources such as RAM and MIPS entirely occupied at the end of the execution. The CFs not only executed their tasks but also executed tasks of other CF as much as possible. The Fig. 10(d) shows that CF4 consumed more energy than other citizen fogs. This indicates that the tasks that other CFs are unable to execute, are executed by the CF4 instead of sending them to the cloud. Only when all the resources are occupied, the tasks are sent to the cloud in no time. The entire cluster of CFs under an MF serves as a larger Fog. Therefore reduces the dependency on the cloud and lessens the delay in real-time services. Fig. 11 shows that in our simulation, the three-layer FCFS Fog



(a) The three-layer Fog Computing Framework in iFogSim



(b) SDFC Framework

Fig. 11. Average successful and delayed task execution rate comparison.

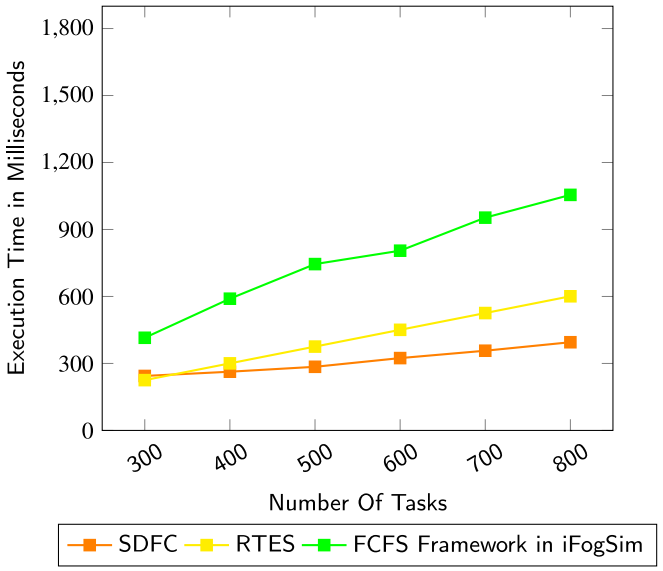


Fig. 12. Application execution time Comparison between SDFC and the three-layer Fog Computing Framework in iFogSim.

Computing Framework in iFogSim executes 42% of tasks successfully and 58% of tasks are delayed. Whereas in SDFC, 80%–85% task is successfully executed, and CFs share their resources for 15%–20% of tasks. Fig. 12 depicts that the application execution time for the three-layer Fog Computing FCFS Framework in iFogSim increases rapidly with the increase in tasks and in RTES, the execution time for a small number of tasks is almost similar to SDFC. However, it starts to take more time than our SDFC as the number of tasks increases. However, for SDFC, the increase is relatively low. From the 100 tasks to 500 tasks, on each iteration of 100, SDFC requires 45%–50% less time than the FCFS Fog Computing Framework in iFogSim.

4.3. Discussion

Our SDFC framework was simulated in iFogSim, implemented the CAA and the LASA algorithm, and compared the results with a three-layer fog computing framework in iFogSim. The algorithms ensure that the waiting time for high priority tasks is minimized

and allocated to the CF with the most available resources. The simulation results in Fig. 10 explains that the available resources are fully utilized, Fig. 11 represents that 15%–20% of tasks are executed on behalf of other tasks, therefore lessening the cloud dependency and increasing the successful execution of tasks up to 80%–85% and finally, Fig. 12 shows that the total execution time decreases by 45%–50%.

5. Conclusion

In this paper, we presented a three-layer framework to enhance resource utilization in a fog computing paradigm. Our SDFC framework includes a Master Fog layer that resides in between the general-purpose fog nodes called Citizen Fogs and the Cloud. The Master Fog is dedicated to bear all the computing overheads to schedule the tasks and allocating the highest priority task to the most eligible CF immediately. The Master Fog uses the CAA algorithm to schedule the tasks based on their priority and the LASA algorithm to sort the citizen fogs according to their available computing ability. Moreover, SDFC assures that the CF layer's resources are utilized adequately, hence reducing the dependency on the cloud. Consequently, not only the task execution time decreases but also the delay in real-time based services reduces. We validated the impact of our SDFC framework and integrated algorithms by simulating in iFogSim. We found that the dependency on the cloud lessens by 15%–20%, and the total execution time reduces by 45%–50%. This reduction in execution time and cloud dependency implicitly decreases delays in real-time services.

Acknowledgment

The authors acknowledge that this research is supported through the Australian Research Council Discovery Project: DP190100314, 'Re-Engineering Enterprise Systems for Microservices in the Cloud'.

References

[1] J. Ni, K. Zhang, X. Lin, X.S. Shen, Securing fog computing for internet of things applications: Challenges and solutions, IEEE Commun. Surveys Tutor. 20 (1) (2017) 601–628.
[2] B.L.R. Stojkoska, K.V. Trivodaliev, A review of internet of things for smart home: Challenges and solutions, J. Cleaner Prod. 140 (2017) 1454–1464.
[3] R. Mahmud, R. Kotagiri, R. Buyya, Fog computing: A taxonomy, survey and future directions, in: Internet of Everything, Springer, 2018, pp. 103–130.
[4] Global number of connected IoT devices 2015-2025 | Statista, https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/ (Accessed on 07 January 2020).
[5] K. Kundhavai, S. Sridevi, IoT and big data-the current and future technologies: A review, Int. J. Comput. Sci. Mob. Comput. 5 (1) (2016) 10–14.
[6] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I.A.T. Hashem, A. Siddiqa, I. Yaqoob, Big IoT data analytics: architecture, opportunities, and open research challenges, IEEE Access 5 (2017) 5247–5261.
[7] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ACM, 2012, pp. 13–16.
[8] P. Brous, M. Janssen, A systematic review of impediments blocking internet of things adoption by governments, in: Conference on E-Business, E-Services and E-Society, Springer, 2015, pp. 81–94.
[9] F. Aliyu, T. Sheltami, E.M. Shakshuki, A detection and prevention technique for man in the middle attack in fog computing, Procedia Comput. Sci. 141 (2018) 24–31.
[10] A.M. Rahmani, T.N. Gia, B. Negash, A. Anzanpour, I. Azimi, M. Jiang, P. Liljeberg, Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach, Future Gener. Comput. Syst. 78 (2018) 641–658.
[11] M. Islam, M. Biswas, M. Mahi, J. Nayeen, M. Whaiduzzaman, et al., LBRP: A resilient energy harvesting noise aware routing protocol for under water sensor networks (UWSNS), Int. J. Found. Comput. Sci. Technol. 8 (2018).
[12] M.J. Islam, M. Mahin, S. Roy, B.C. Debnath, A. Khatun, Distblacknet: A distributed secure black SDN-IoT architecture with NFV implementation for smart cities, in: 2019 International Conference on Electrical, Computer and Communication Engineering, ECCE, IEEE, 2019, pp. 1–6.
[13] A. Barros, U. Kylau, Service delivery framework - an architectural strategy for next-generation service delivery in business network, in: 2011 Annual SRII Global Conference, 2011, pp. 47–58, http://dx.doi.org/10.1109/SRII.2011.15.
[14] G. Decker, A. Barros, F.M. Kraft, N. Lohmann, Non-desynchronizable service choreographies, in: A. Bouguettaya, I. Krueger, T. Margaria (Eds.), Service-Oriented Computing – ICSOC 2008, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 331–346.
[15] A.P. Barros, A. Grosskopf, Process model control flow with multiple synchronizations, 2013, Google Patents, US Patent 8, 429, 653.
[16] F. Al-Doghman, Z. Chaczko, A.R. Ajayan, R. Klempous, A review on fog computing technology, in: 2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC, IEEE, 2016, pp. 001525–001530.
[17] M. Aazam, S. Zeadally, K.A. Harras, Fog computing architecture, evaluation, and future research directions, IEEE Commun. Mag. 56 (5) (2018) 46–52.
[18] L.F. Bittencourt, J. Diaz-Montes, R. Buyya, O. Rana, M. Parashar, Mobility-aware application scheduling in fog computing, IEEE Cloud Comput. 4 (2017) 26–35, http://dx.doi.org/10.1109/MCC.2017.27.
[19] ZDNet, Real-time data generation by 2025, 2020, https://www.zdnet.com/article/by-2025-nearly-30-percent-of-data-generated-will-be-real-time-idc-says/.
[20] J. Liu, J. Li, L. Zhang, F. Dai, Y. Zhang, X. Meng, J. Shen, Secure intelligent traffic light control using fog computing, Future Gener. Comput. Syst. 78 (2018) 817–824.
[21] S. Sukode, S. Gite, Vehicle traffic congestion control & monitoring system in IoT, Int. J. Appl. Eng. Res. 10 (8) (2015) 19513–19523.
[22] Q. Fan, N. Ansari, Towards workload balancing in fog computing empowered IoT, IEEE Trans. Netw. Sci. Eng. (2018).
[23] M. Whaiduzzaman, A. Gani, A. Naveed, PEFC: Performance enhancement framework for cloudlet in mobile cloud computing, in: 2014 IEEE International Symposium on Robotics and Manufacturing Automation, ROMA, IEEE, 2014, pp. 224–229.
[24] B. Sikdar, A study of the environmental impact of wired and wireless local area network access, IEEE Trans. Consum. Electron. 59 (1) (2013) 85–92.
[25] S.K. Datta, C. Bonnet, J. Haerri, Fog computing architecture to enable consumer centric internet of things services, in: 2015 International Symposium on Consumer Electronics, ISCE, IEEE, 2015, pp. 1–2.
[26] A. Gyrard, S.K. Datta, C. Bonnet, K. Boudaoud, Integrating machine-to-machine measurement framework into oneM2M architecture, in: 2015 17th Asia-Pacific Network Operations and Management Symposium, APNOMS, IEEE, 2015, pp. 364–367.
[27] I. Stojmenovic, Fog computing: A cloud to the ground support for smart things and machine-to-machine networks, in: 2014 Australasian Telecommunication Networks and Applications Conference, ATNAC, IEEE, 2014, pp. 117–122.

[28] E. Yigitoglu, M. Mohamed, L. Liu, H. Ludwig, Foggy: a framework for continuous automated IoT application deployment in fog computing, in: 2017 IEEE International Conference on AI & Mobile Services, AIMS, IEEE, 2017, pp. 38–45.
[29] M. Whaiduzzaman, S.R. Tuly, N. Haque, M. Hossain, A. Barros, Credit based task scheduling process management in fog computing, in: Pacific Asia Conference on Information Systems (PACIS) Proceedings, 2020, pp. 232.
[30] D. Santoro, D. Zozin, D. Pizzolli, F. De Pellegrini, S. Cretti, Foggy: a platform for workload orchestration in a fog computing environment, in: 2017 IEEE International Conference on Cloud Computing Technology and Science, CloudCom, IEEE, 2017, pp. 231–234.
[31] X. Masip-Bruin, E. Marín-Tordera, G. Tashakor, A. Jukan, G.-J. Ren, Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems, IEEE Wirel. Commun. 23 (5) (2016) 120–128.
[32] J. Das, A. Mukherjee, S. Ghosh, R. Buyya, Spatio-fog: A green and timeliness-oriented fog computing model for geospatial query resolution, Simul. Model. Pract. Theory 100 (2019) 102043, http://dx.doi.org/10.1016/j.simpat.2019.102043.
[33] P. Habibi, S. Baharlooei, M. Farhoudi, S. Kazemian, S. Khorsandi, Virtualized SDN-Based End-to-End Reference Architecture for Fog Networking, 2018, pp. 61–66, http://dx.doi.org/10.1109/WAINA.2018.00064.
[34] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, R. Boutaba, Network function virtualization: State-of-the-art and research challenges, IEEE Commun. Surv. Tutor. 18 (1) (2016) 236–262.
[35] E. Haleplidis, K. Pentikousis, S. Denazis, J. Salim, D. Meyer, O. Koufopavlou, RFC 7426: Software-defined networking (SDN): Layers and architecture terminology, IRTF (2015).
[36] K. Velasquez, D. Perez Abreu, D. Goncalves, L.F. Bittencourt, M. Curado, E. Monteiro, E. Madeira, Service Orchestration in Fog Environments, 2017, pp. 329–336, http://dx.doi.org/10.1109/FiCloud.2017.49.
[37] M. Brito, S. Hoque, T. Magedanz, R. Steinke, A. Willner, D. Nehls, O. Keils, F. Schreiner, A Service Orchestration Architecture for Fog-Enabled Infrastructures, 2017, pp. 127–132, http://dx.doi.org/10.1109/FMEC.2017.7946419.
[38] X.-Q. Pham, E.-N. Huh, Towards task scheduling in a cloud-fog computing system, in: 2016 18th Asia-Pacific Network Operations and Management Symposium, APNOMS, IEEE, 2016, pp. 1–4.
[39] T. Choudhari, M. Moh, T.-S. Moh, Prioritized task scheduling in fog computing, ACMSE '18 Proceedings of the ACMSE 2018 Conference (2018) 1–8, http://dx.doi.org/10.1145/3190645.3190699.
[40] S. Agarwal, S. Yadav, A. Yadav, An efficient architecture and algorithm for resource provisioning in fog computing, Int. J. Inform. Eng. Electron. Bus. 8 (2016) 48–61, http://dx.doi.org/10.5815/ijieeb.2016.01.06.
[41] M. Verma, N. Bhardwaj, A.K. Yadav, Real time efficient scheduling algorithm for load balancing in fog computing environment, Int. J. Inf. Technol. Comput. Sci. 8 (4) (2016) 1–10.
[42] H.-M. Chu, S.-W. Yang, P. Pillai, Y.-K. Chen, Scheduling in visual fog computing: NP-completeness and practical efficient solutions, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
[43] W. Lin, C. Zhu, J. Li, B. Liu, H. Lian, Novel algorithms and equivalence optimisation for resource allocation in cloud computing, Int. J. Web Grid Serv. 11 (2) (2015) 193–210.
[44] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, J. Parallel Distrib. Comput. 59 (2) (1999) 107–131.
[45] T. Brauny, H. Siegely, N. Becky, et al., A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems, J. Parallel Distrib. Comput. 61 (6) (2001) 810–837.
[46] H. Wang, J. Gong, Y. Zhuang, H. Shen, J. Lach, Healthedge: Task scheduling for edge computing with health emergency and human behavior consideration in smart homes, in: 2017 IEEE International Conference on Big Data (Big Data), IEEE, 2017, pp. 1213–1222.
[47] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, S. Davy, Design and evaluation of algorithms for mapping and scheduling of virtual network functions, in: Proceedings of the 2015 1st IEEE Conference on Network Softwarization, NetSoft, IEEE, 2015, pp. 1–9.
[48] M. Al-khafajiy, T. Baker, H. Al-Libawy, A. Waraich, C. Chalmers, O. Alfandi, Fog computing framework for internet of things applications, in: 2018 11th International Conference on Developments in ESystems Engineering, DeSE, IEEE, 2018, pp. 71–77.
[49] D. Tychalas, H. Karatza, A scheduling algorithm for a fog computing system with bag-of-tasks jobs: Simulation and performance evaluation, Simul. Model. Pract. Theory 98 (2019) 101982, http://dx.doi.org/10.1016/j.simpat.2019.101982.
[50] M.A.A. Mostafa, A.M.K. Mohammad, Cognitive management framework for fog computing in IoT case study: Traffic control system, in: 2017 8th International Conference on Information Technology, ICIT, IEEE, 2017, pp. 875–882.
[51] M.B. Yassein, M.Q. Shatnawi, S. Aljwarneh, R. Al-Hatmi, Internet of things: Survey and open issues of MQTT protocol, in: 2017 International Conference on Engineering & MIS, ICEMIS, IEEE, 2017, pp. 1–6.
[52] D.B. LD, P.V. Krishna, Honey bee behavior inspired load balancing of tasks in cloud computing environments, Appl. Soft Comput. 13 (5) (2013) 2292–2303.
[53] M. Whaiduzzaman, M. Sookhak, A. Gani, R. Buyya, A survey on vehicular cloud computing, J. Netw. Comput. Appl. 40 (2014) 325–344.
[54] University of Minnesota, The performance equation, 2020, https://www.d.umn.edu/~gshute/arch/performance-equation.xhtml.
[55] H. Gupta, A. Dastjerdi, S. Ghosh, R. Buyya, iFogSim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments, Softw. - Pract. Exp. (2016) http://dx.doi.org/10.1002/spe.2509.
[56] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers, Concurr. Comput.: Pract. Exper. 24 (2012) http://dx.doi.org/10.1002/cpe.1867.