

Efficient and dynamic scaling of fog nodes for IoT devices

Said El Kafhali¹ · Khaled Salah²

Published online: 5 June 2017

© Springer Science+Business Media New York 2017

Abstract It is predicted by the year 2020, more than 50 billion devices will be connected to the Internet. Traditionally, cloud computing has been used as the preferred platform for aggregating, processing, and analyzing IoT traffic. However, the cloud may not be the preferred platform for IoT devices in terms of responsiveness and immediate processing and analysis of IoT data and requests. For this reason, fog or edge computing has emerged to overcome such problems, whereby fog nodes are placed in close proximity to IoT devices. Fog nodes are primarily responsible of the local aggregation, processing, and analysis of IoT workload, thereby resulting in significant notable performance and responsiveness. One of the open issues and challenges in the area of fog computing is efficient scalability in which a minimal number of fog nodes are allocated based on the IoT workload and such that the SLA and QoS parameters are satisfied. To address this problem, we present a queuing mathematical and analytical model to study and analyze the performance of fog computing system. Our mathematical model determines under any offered IoT workload the number of fog nodes needed so that the QoS parameters are satisfied. From the model, we derived formulas for key performance metrics which include system response time, system loss rate, system throughput, CPU utilization, and the mean number of messages request. Our analytical model is cross-validated using discrete event simulator simulations.

✉ Said El Kafhali
said.elkafhali@uhp.ac.ma

Khaled Salah
khaled.salah@kustar.ac.ae

¹ Computer, Networks, Mobility and Modeling Laboratory, National School of Applied Sciences, Hassan 1st Univ, Settat, Morocco

² Electrical and Computer Engineering Department, Khalifa University of Science, Technology and Research (KUSTAR), Abu Dhabi, UAE

Keywords Internet of things · Fog computing · Edge computing · Cloud computing · Queuing theory · Performance modeling and analysis

1 Introduction

Cloud computing is a new IT computing infrastructure that has the ability to host and run software services and applications at a fraction of the cost. Many of today's cloud providers such as Google, Amazon Web Services (AWS), Microsoft, IBM, and others lead this industry by offering innovative services to cater client needs. From client perspective, the cloud-hosted applications, services, and data can be accessed in ubiquitous manner. Meanwhile, the deployment of smart connected devices (aka IoT devices) has been exponentially on the rise. By 2020, Cisco conservatively estimates that 50 billion IoT devices will be connected to the Internet [1].

Ample of research has been conducted on leveraging cloud computing technologies to support and facilitate IoT [2,3]. The integration of IoT and the cloud has brought about a new paradigm of pervasive and ubiquitous computing, called Cloud of Things (CoT) [4]. CoT is an IoT connected product management solution that enables its clients to connect any device to any cloud data center (CDC). The increasing number of IoT devices will inevitably produce a huge amount of data, which have to be processed, stored, and properly accessed seamlessly and ubiquitously by end clients. The IoT devices often have limited computing and processing capacity and are not able to perform sophisticated processing and storing large amounts of data [5]. Hence, cloud computing seems to be the best alternative to meet the requirements of IoT scenarios. However, the cloud platform introduces obvious concerns and limitations in terms of responsiveness, latency, and performance in general for processing and accessing IoT traffic data. It is time-consuming especially for large data sets to travel back and forth between client and cloud [6].

For addressing the limitations of CoT architecture, a new and promising computing paradigm called fog or edge computing has recently been advocated [7,8]. In fact, many telecom network companies have started building a fog computing system at the network edge to cater the emerging bandwidth hungry applications and to minimize its operational cost and applications response time [9]. Fog computing (FC) architecture [10] is a distributed computing paradigm that empowers the network edge devices at different hierarchical layers with many degrees of computational and storage capability [7]. Edge computing (EC) enables clients to access IT and cloud computing services at close proximity, thereby enriching client's satisfaction and improving Quality of Experience (QoE). In other words, the EC allows computation and services to be performed at the edge. Consequently, many improvements will come about which include improving QoS (in terms of lower latency and higher throughput), improving scalability and resource efficiency, and easing the way to implement authentication and authorization to IoT devices [11]. In the majority of the time, EC can perform the processing of workload and data generated by IoT devices at the edge, i.e., without forwarding such workload or data to the CDC. However, the cloud platform is needed for sophisticated processing (such as data analytics), which entail heavy and complex compute processing and storage capacities, and relies on historical data stored for a

long period of time. It should be noted clearly that, in the context of IoT, fog computing is not a substitute of cloud computing, rather these two technologies complement one another. The complementary functions of cloud and fog computing enable the clients to experience a new breed of computing technology that serves the requirements of the real-time processing and low latency for IoT applications running at the EC, and also supports complex analysis and long-term storage of data at the CDC [12].

Performance modeling and analysis have been and continued to be of high theoretical and practical importance in research, development, and optimization of computer applications and communication systems [13]. This includes a broad of research activities from the use of more empirical methods through the use of more sophisticated mathematical approaches to simulation [14]. These have played an important role in understanding critical problems, development, management, and planning of complex systems. Queuing theory has been used widely in studying and modeling the performance and QoS for different ICT systems [15]. By using queuing modeling, one can then derive key system performance and QoS parameters which may include mean response time, mean request queuing length, mean waiting time, mean throughput, CPU utilization, and blocking probability [16].

Dynamic scaling in the context of IoT and fog nodes is the ability to scale up and down resources according to the incoming IoT workload. So if the aggregated IoT workload goes up, more fog nodes have to be allocated to meet such increase, and conversely, if IoT workload goes down, the number of provisioned fog nodes has to decrease. However, the number of fog nodes allocated has to always satisfy an SLA parameter. In a fog computing environment, the primary challenge is to use minimal EC resources at any given time in order to satisfy the SLA performance requirements. As the IoT workload changes over time, it is important to dynamically provision the minimal number of ECs to satisfy the agreed-on SLA requirements. Static resource allocation increases the cost of running the IoT services. Allocating more EC than required to satisfy the SLA will result in over-provisioning and higher cost to the IoT provider. On the other hand, using fewer ECs than required will lead to under-provisioning whereby the SLA performance requirements are violated. Hence, dynamic resources allocations are essential in order to avoid the over-provisioning and under-provisioning situations. However, it is critical for the IoT provider to determine the correct number of ECs needed to handle the presented IoT workload and at the same time be able to satisfy the SLA requirements (say response time). In this paper, we propose an analytical model and derive formulas and also show examples of how to dynamically and efficiently scale EC nodes such that any violation to SLA requirements is avoided.

In this paper, we present an analytical model based on queuing theory to capture the dynamicity of IoT systems and study their performance. Specifically, we show how our model can be used to model and estimate the QoS parameters for IoT devices using a combination of fog and cloud architecture. We also show how our model can be used for dynamic scalability of fog/edge nodes. Our proposed analytical model addresses and provides answers to these specific three main questions: (1) Given the IoT workload and the available edge computing nodes, what is the QoS offered to users? (2) Given the IoT workload and a target QoS, how many edge computing nodes

are needed? and (3) Given the available edge computing nodes and a target QoS, what is the maximum IoT workload supported by the system?

The main contributions of this work can be summarized as follows:

- A queuing analytical model is presented to capture the dynamics and behavior of fog nodes of IoT devices. The analytical model is composed of three concatenated sub-systems including edge nodes, cloud gateway, and CDC queuing models.
- Mathematical formulas are derived from the analytical model for key performance measures.
- Numerical examples are given to show how our analytical model can be used to predict the performance of fog computing system, and also to determine the required number of edge nodes needed under variable IoT workload conditions.
- Discrete event simulations are used to cross-check and validate the accuracy of our analytical models.

The rest of the paper is organized as follows: A description of fog computing architecture is presented in Sect. 2. The proposed fog computing model is presented in Sect. 3. Section 4 presents analytical model for the proposed model. Section 5 presents numerical and simulation results. Section 6 summarizes the related work. Finally, Sect. 7 includes concluding remarks and future works.

2 The system architecture of fog computing

In this section, we present the system architecture of the three-tier FC as shown in Fig. 1, and we summarize the details related to each tier. As started earlier, FC is a virtualized architecture that provides compute, storage, and networking services between traditional CDC and end IoT devices, typically located at the edge of network [17]. FC essentially involves components of an application running both in the CDC and in EC between IoT devices and the cloud that is, in smart gateways, routers, or dedicated fog devices. It supports mobility, communication protocols, computing resources, distributed data analytics, interface heterogeneity, and cloud integration to address requirements of applications that need low latency with a wide and dense geographical distribution [18]. The majority of traffic and data generated by IoT devices are not processed at the CDC, since the storage and processing capabilities can be too far from the IoT devices. The needs of geo-spatially distribution of resources, real-time communication, and incorporation with large networks are handled by FC. Through the FC intermediary layer, part of processing is done by EC closer to IoT devices, resulting in less response time and bandwidth usage.

As shown in Fig. 1, the FC system contains three distinct physical tiers, namely Tier 1, Tier 2, and Tier 3. Tier 1 is also known as the IoT devices tier. It is the bottom tier where the physical sensors IoT devices are placed [19]. At this tier, the IoT devices generate traffic and workload to the system backend for further processing by hosted applications at either the fog or cloud tier. The traffics transmitted by the IoT devices are received by the access points present at the border of the fog tier [20]. These traffics are then redirected to the fog layer for a possible processing by the EC devices or redirected by the EC to cloud layer through cloud gateway.

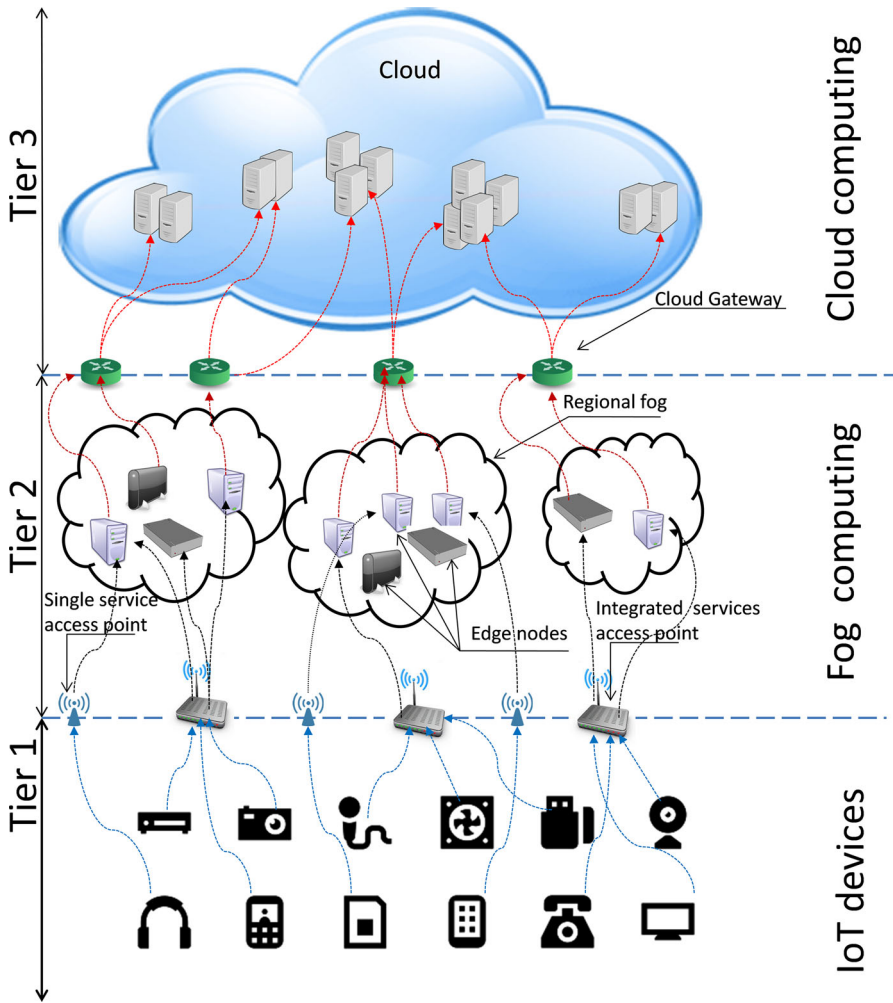


Fig. 1 Three-tier architecture of a typical IoT system

Tier 2 is the FC tier, and it composed by the EC devices. These EC devices are connected to the cloud through cloud gateway and are responsible for sending traffic to the CDC on a periodic basis. The EC nodes have limited storage capacity that allows them to store the received data temporarily for analysis and then send the source devices the needful feedbacks [21]. Unlike the traditional cloud platform, in FC all real-time analysis and latency-sensitive applications take place at the fog layer itself. These real-time systems are used to provide context-aware services to the connected IoT devices, thereby enriching client’s satisfaction and enhancing QoE. The EC paradigm is a federation of resource-poor devices [e.g., sensors, radio frequency identification (RFID)], human-controlled devices (e.g., smartphone, tablets), stable networking devices (e.g., switches, routers), and resource-rich machines (e.g., cloudlets) [17,22].

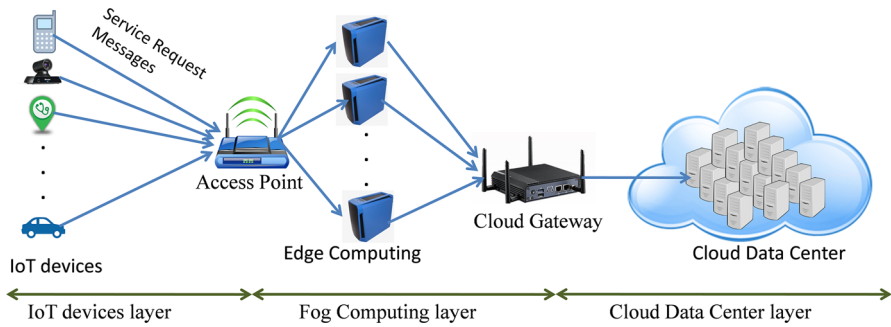


Fig. 2 Architectural model for request flow and processing

Tier 3 is the cloud computing layer. The key components in this layer are the data centers (DCs), which are used to provide the required computing capability and storage to the clients and applications based on the pay-as-you-go model. A DC is a facility consisting of physical servers (PSs), storage and network devices (e.g., switches, routers, and cables), power distribution systems, and cooling systems [23]. Indeed, large DCs of Google, Microsoft, Yahoo, and Amazon contain tens of thousands of PSs [24]. Each PS is provisioned as many virtual machines (VMs) in real time subject to availability and agreement on service levels according to the client service-level agreement (SLA) document.

3 Fog computing architectural model

3.1 Model description

In this section, we present our fog computing architectural model by defining its three-tier network structure, as depicted in Fig. 2. The first tier is the bottom layer encompasses all IoT devices, which are responsible for sensing of a multitude of events and transmitting the raw sensed data to its immediate upper layer. We assume that the total number of IoT devices is constant over time and equal to X end clients. The access point acts as a connected point (through wireless or wired links) between IoT devices and the EC nodes. That access points receive incoming traffic from end clients. These IoT device messages get aggregated at the access points (located in close proximity of the IoT devices) and subsequently get sent to the edge nodes. The middle tier, also known as the fog computing layer, comprises edge nodes intelligent enough to process, compute, and temporarily store the received information and forward other remaining requests or workload to the cloud tier for further processing or storage. These EC devices are connected to the CDC and are responsible for sending data to the cloud through a cloud gateway (CG). The last tier is the cloud data center layer. This layer constitutes a large data center with a collection of PSs, where VMs run on the top of PS capable of processing and storing an enormous amount of data.

As depicted in Fig. 3, the arrival process of incoming client message is a Poisson process with an aggregate arrival rate λ . It is to be noted that his aggregate rate is the total arrival rate of the individual IoT devices and can be expressed as

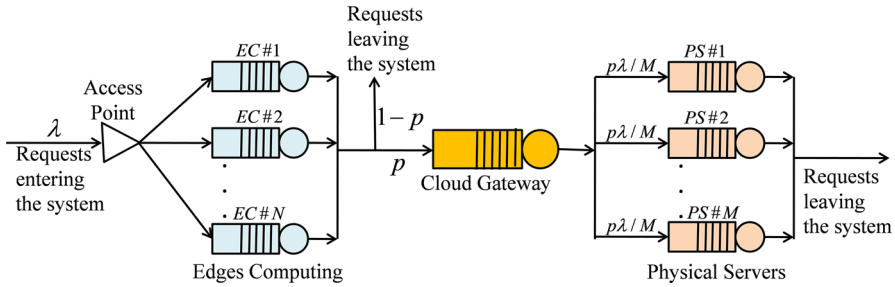


Fig. 3 Queuing model of fog computing system

$$\lambda = \sum_{i=1}^X \lambda_i \tag{1}$$

where i is an index of a total of X IoT devices. Please note we assume the individual arrival rates λ_i follow a Poisson process, and this makes the aggregate arrival rate λ also Poisson. We also assume that the incoming messages requests are being serviced following first-come first-served (FCFS) policy.

There are N edges computing and M servers in the system. The processing times of the EC are independent and identically distributed exponential random variables with mean rate $1/\mu_{EC}$. First, the incoming message which may carry data or requests is served by the EC. After being processed by the EC node, the message is either forwarded to CDC for further service with probability p or completes the service and departs from the system with probability $1 - p$. Each EC is modeled as an $M/M/1/C$ queue. The maximum number of messages in each EC is C , which implies a maximum queue length of $C - 1$. An arriving message enters the queue if it finds less than C request messages in the EC and is lost otherwise. The CG has load balancer functionality and has been able to dispatch traffic from EC evenly among the VMs in CDC. The CG is modeled as an $M/M/1$ queue. The $M/M/n/K$ queuing model is employed to characterize each PS. The service times of the VMs are independent and identically distributed exponential random variables with rate $1/\mu_{VM}$. Table 1 describes system input parameters used in our modeling.

3.2 Limitations

For our analysis, we assumed that the aggregate message arrival rate of all IoT devices follows a Poisson arrival and the service times for the individual server or EC node are exponentially distributed. These assumptions may not necessarily always hold. In fact, the incoming request behavior in such system must rely on more complex models than a simple Poisson process (such as self-similar behavior, business behavior, and long-range dependency) [25,26]. Also, the distribution of services times of servers and EC nodes may not always be exponential. It is worth noting that an analytical solution becomes intractable when considering non-Poisson arrivals and when also considering general service times. On the other hand, assuming Poisson arrivals and exponential

Table 1 Key system input parameters

Parameters	Descriptions
λ_i	Aggregate message arrival rate from i th IoT device
λ	Aggregate message arrival rate of all IoT devices
$1/\mu_{EC}$	Mean EC service time
$1/\mu$	Mean GC service time
$1/\mu_{VM}$	Mean VM service time
p	The probability that a message request forwarded to CDC from an EC node
$1 - p$	The probability that a message request completes their service in EC and departs from the system
X	Maximum number of IoT devices
C	Maximum number of message requests in each EC
K	Maximum number of message requests in each PS
N	Number of EC nodes in the system
M	Number of PSs in the data center
n	Number of VMs in each PS

service time has used in the literature and can provide adequate approximation of real systems [27–30].

4 Analytical modeling

4.1 Edge computing model

As shown in Fig. 3, the system will contain N EC nodes and each of them is modeled as an $M/M/1/C$ queue. In an $M/M/1/C$ queuing system, the maximum number of message requests in the system is C , which implies a maximum queue length of $C - 1$. An arriving message request enters the queue if it finds fewer than C message requests in the system and is lost otherwise. Since there are no more than C message requests in the EC system (finite), the system is stable for all values of λ and μ_{EC} . Let π_k^j denote the equilibrium probability that there are k message requests in the j th EC ($j = 1, \dots, N$). Using the balanced equations and the normalization condition [13, 16]:

$$\begin{cases} \pi_0^j \lambda = \pi_1^j \mu_{EC} \\ \pi_1^j (\lambda + \mu_{EC}) = \pi_0^j \lambda + \pi_2^j \mu_{EC} \\ \dots \\ \pi_k^j (\lambda + \mu_{EC}) = \pi_{k-1}^j \lambda + \pi_{k+1}^j \mu_{EC}, \quad k = 2, \dots, C - 1 \\ \pi_C^j \mu_{EC} = \pi_{C-1}^j \lambda \\ \sum_{k=0}^C \pi_k^j = 1 \end{cases} \quad (2)$$

We obtain the steady-state probability of k message requests in the j th EC as below

$$\pi_k^j = \begin{cases} \frac{(1-a)a^k}{1-a^{C+1}}, & a \neq 1 \\ \frac{1}{C+1}, & a = 1 \end{cases} \tag{3}$$

where $a = \lambda/\mu_{EC}$ denotes the offered load in each EC server.

We can obtain the probability that a message request arriving at j th EC will find k message requests ($0 \leq k < C$) already there is given by

$$P_k^j = \begin{cases} \frac{(1-a)a^k}{1-a^C}, & a \neq 1 \\ \frac{1}{C}, & a = 1 \end{cases} \tag{4}$$

The overflow probability that an arriving message request at j th EC will see $B \leq k < C$ message requests already in the queue is

$$P_B^j = \sum_{k=B}^{C-1} P_k^j = \frac{a^B - a^C}{1 - a^C}, \quad a \neq 1 \tag{5}$$

The effective rate of message arrivals, λ_e , to the service in j th EC is given by

$$\lambda_e = \lambda(1 - \pi_C^j) \tag{6}$$

The rate of loss can be obtained as follows

$$P_{\text{loss}}^j = \lambda\pi_C^j = \lambda \frac{1 - a}{1 - a^{C+1}} a^C, \quad a \neq 1 \tag{7}$$

The mean throughput service \bar{X}_{EC}^j of j th EC is given by

$$\bar{X}_{EC}^j = \lambda(1 - P_{\text{loss}}) = \lambda \frac{1 - a^C}{1 - a^{C+1}}, \quad a \neq 1 \tag{8}$$

The EC server utilization is

$$U_{EC} = \frac{\bar{X}_{EC}^j}{\mu_{EC}} = a \frac{1 - a^C}{1 - a^{C+1}}, \quad a \neq 1 \tag{9}$$

The mean number of message requests in the j th EC is

$$\bar{E}_{EC}^j = \sum_{k=1}^C k\pi_k^j = \begin{cases} \frac{a}{1-a} \frac{1-(C+1)a^C+Ca^{C+1}}{1-a^{C+1}}, & a \neq 1 \\ \frac{C}{2}, & a = 1 \end{cases} \tag{10}$$

The average busy servers is the number of message request in service is expected as

$$\bar{L}_{EC}^j = 1 - \frac{1-a}{1-a^{C+1}}, \quad a \neq 1 \quad (11)$$

The mean number of message requests waiting in the j th EC is

$$\bar{M}_{EC}^j = \bar{E}_{EC}^j - \bar{L}_{EC}^j \quad (12)$$

The mean waiting time of message requests at the j th EC as

$$\bar{W}_{EC}^j = \frac{\bar{M}_{EC}^j}{\lambda} \quad (13)$$

Finally, the mean response time of message requests at the j th EC as

$$\bar{T}_{EC}^j = \frac{\bar{E}_{EC}^j}{\bar{X}_{EC}^j} = \frac{1}{\mu_{EC} - 1} - \frac{C\lambda^C}{\mu_{EC}^{C+1} - \lambda^C}, \quad a \neq 1 \quad (14)$$

We deduce formulas for key performance metrics related to the edge computing sub-system. First, the rate of loss due to lack of space in all EC queues

$$P_{\text{loss}} = \sum_{j=1}^N P_{\text{loss}}^j \quad (15)$$

The mean throughput service is given by

$$\bar{X}_{EC} = \sum_{j=1}^N \bar{X}_{EC}^j \quad (16)$$

The mean number of message requests in the EC sub-system is

$$\bar{E}_{EC} = \sum_{j=1}^N \bar{E}_{EC}^j \quad (17)$$

The mean waiting time of message requests at the EC sub-system is

$$\bar{W}_{EC} = \sum_{j=1}^N \bar{W}_{EC}^j \quad (18)$$

Finally, the mean response time of message requests is

$$\bar{T}_{EC} = \frac{\bar{E}_{EC}}{\bar{X}_{EC}}. \quad (19)$$

4.2 Cloud gateway model

In a CDC, resources are composed of VMs, which are deployed on PSs. It is advantageous for the cloud provider to accept and fulfill as many incoming message requests as possible in order to keep its utilization of resources and maximizes return on investment. To this end, the CG in our system has a load balancing functionality, to balance traffic received from EC and distributed it evenly among the VMs in CDC. Load balancer is an important technique for online resources management to control the data flows from different applications. In addition, the choice of an appropriate VM is very important to enhance service functionality, increase throughput, avoid network overloading, reduce response time, and minimize cost. To avoid the rejection of traffic between FC layer and cloud layer, the CG is modeled as an $M/M/1$ queue. According to Burke's theorem [31], the output of a finite $M/M/1/C$ queue with an input parameter λ is a Poisson process with the same input parameter λ . Since the message request enters the system through the EC model, then they move on to the CG queue with probability p . Hence, the message request arrivals follow a Poisson process with arrival rate $p\lambda$. The processing times of the CG are independent and identically distributed exponential random variables with mean rate $1/\mu$. We assume that $p\lambda < \mu$, so the underlying continuous-time Markov chain (CTMC) is ergodic and hence the queuing system is stable. Using the global balance equations and the normalization condition, we obtain the steady-state probability of the system being empty

$$\pi_0 = \frac{1}{1 + \sum_{k=1}^{\infty} \left(\frac{p\lambda}{\mu}\right)^k} = \frac{1}{1 + \frac{p\lambda/\mu}{1-p\lambda/\mu}} = 1 - \frac{p\lambda}{\mu} \quad (20)$$

For the steady-state probability that there are k message requests in the system, we get

$$\pi_k = \pi_0 \left(\frac{p\lambda}{\mu}\right)^k = \left(1 - \frac{p\lambda}{\mu}\right) \cdot \left(\frac{p\lambda}{\mu}\right)^k, \quad k \geq 0 \quad (21)$$

With the utilization $\eta = p\lambda/\mu$, we can obtain

$$\pi_k = (1 - \eta)\eta^k, \quad k \geq 0 \quad (22)$$

Now, we can derive the key performance indicators of CG system as follows. First, the mean message requests are obtained as

$$\bar{E}_{CG} = \sum_{k=1}^{\infty} k\pi_k = (1 - \eta) \sum_{k=1}^{\infty} k\eta^k = \frac{\eta}{1 - \eta} \quad (23)$$

The mean message requests waiting in CG queue are obtained as

$$\bar{Q}_{CG} = \sum_{k=1}^{\infty} k\pi_{k+1} = \frac{\eta^2}{1 - \eta} \quad (24)$$

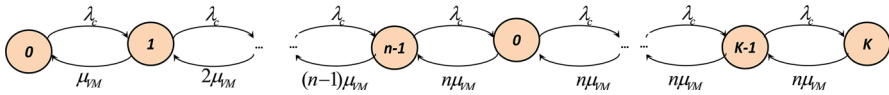


Fig. 4 Continuous-time Markov chain for new request within a single PS

The throughput of the CG system can be obtained from

$$\bar{X}_C = p\lambda(1 - \pi_C) \tag{25}$$

The CG queue is being utilized whenever it is non-empty; in other words the utilization of CG is as follows

$$\bar{U}_{CG} = 1 - \pi_0 = \eta \tag{26}$$

Finally, by using Little’s theorem [32], we get the following for the mean response time and the mean waiting time

$$\bar{T}_{CG} = \frac{\bar{E}_{CG}}{p\lambda} = \frac{1/\mu}{1 - \eta}, \quad \bar{W}_{CG} = \frac{\bar{Q}_{CG}}{p\lambda} = \frac{\eta/\mu}{1 - \eta}. \tag{27}$$

4.3 Cloud data center model

The model we use to describe each PS in CDC corresponds to $M/M/n/K$ queuing system. We assume there are M homogenous PSs in the CDC with n VMs at each PS, and K is the maximum number of message requests in each PS. The client messages are evenly distributed by the CG server to each PS with the same probability $1/M$. Consequently, the incoming client message is a Poisson process with an aggregate arrival rate $\lambda_c = p\lambda/M$. We assume that all VMs are homogeneous service. Therefore, the processing server time of each VM is exponentially distributed with mean service time server $1/\mu_{VM}$. Let us define the state of a PS queue as the total number of VMs in the PS [33]. We assume that there are no waiting buffers in the VMs. Figure 4 exhibits the transition diagram for the new message request in a single PS.

Let $\pi_i(k)$ denote the steady-state probability of having k traffic requests in PS _{i} ($i = 1, 2, \dots, M$). Using the balanced Eqs. [14, 15], we find that

$$\pi_i(k) = \begin{cases} \frac{\pi_0(\lambda_c)^k}{k!(\mu_{VM})^k}, & \forall k < n \\ \frac{\pi_0(\lambda_c)^k}{n!n^{k-n}(\mu_{VM})^k}, & \forall k \geq n \end{cases} \tag{28}$$

where π_0 is given by the normalization as

$$\pi_0 = \left[1 + \frac{(n\rho)^n(1 - \rho^{K+1-n})}{n!(1 - \rho)} + \sum_{i=1}^{n-1} \frac{(n\rho)^i}{i!} \right]^{-1} \tag{29}$$

where ρ denotes the offered load and is expressed as

$$\rho = \frac{\lambda_c}{n\mu_{VM}} \quad (30)$$

The effective rate of message arrivals, λ_{eff} , to the service is given by

$$\lambda_{eff} = \lambda_c(1 - \pi_i(K)) \quad (31)$$

The throughput, \bar{X}_C^i , which is defined as the mean number of message requests serviced during a time unit is expressed as follows

$$\bar{X}_C^i = \rho(1 - \pi_i(K)) \quad (32)$$

We then deduce the key performance measures as follows. First, the rate of loss can be obtained as follows

$$v^i = \lambda_c \pi_i(K) \quad (33)$$

The CPU utilization of each VM instance can be expressed as follows

$$U_{VM} = \frac{\lambda_{eff}}{n\mu_{VM}} = \rho(1 - \pi_i(K)) \quad (34)$$

We compute \bar{E}_C^i , the mean number of message requests in the i th PS as

$$\bar{E}_C^i = \sum_{j=1}^K j\pi_i(j) \quad (35)$$

The mean number of message requests waiting in the i th PS can be obtained from

$$\bar{E}_W^i = \sum_{j=n+1}^K (j - n)\pi_i(j) \quad (36)$$

Lastly, we use Little's formula [32] to obtain the mean response time and mean waiting time in the i th PS as follows

$$\bar{T}_R^i = \frac{\bar{E}_C^i}{\lambda_c(1 - \pi_i(K))}, \quad \bar{T}_W^i = \frac{\bar{E}_W^i}{\lambda_c(1 - \pi_i(K))} \quad (37)$$

We deduce the performance metrics related to the CDC sub-system. First, the rate of loss due to lack of space in all PM queues

$$v = \sum_{i=1}^M v^i \quad (38)$$

The mean number of message requests as

$$\bar{E}_C = \sum_{i=1}^M \bar{E}_C^i \quad (39)$$

The mean number of message requests waiting in the CDC sub-system can be obtained from

$$\bar{E}_W = \sum_{i=1}^M \bar{E}_W^i \quad (40)$$

The throughput of the CDC system can be obtained from [34] as follows

$$\bar{X}_C = \sum_{i=1}^M \bar{X}_C^i \quad (41)$$

Lastly, the mean response time and mean waiting time in the CDC sub-system as follows

$$\bar{T}_R = \frac{\bar{E}_C}{\bar{X}_C}, \quad \bar{T}_W = \frac{\bar{E}_W}{\bar{X}_C}. \quad (42)$$

4.4 Performance metrics for the overall system

Based on analyzing the three concatenated queuing sub-systems, we derive now key performance formulas for the entire queuing system as follows. First, the mean response time is the sum of the response time of the three queuing models. The system throughput can be obtained from

$$\bar{X} = \bar{X}_{EC} + \bar{X}_{CG} + \bar{X}_C \quad (43)$$

The mean response time equation can be formulated as

$$\bar{T} = \bar{T}_{EC} + \bar{T}_{CG} + \bar{T}_R \quad (44)$$

The mean waiting time for a message request can be expressed as

$$\bar{W} = \bar{W}_{EC} + \bar{W}_{CG} + \bar{T}_W \quad (45)$$

Since the CG is an infinite capacity queuing model, the rate of loss of the system is related to the finite capacity of the EC sub-system and the finite capacity of CDC sub-system.

$$P = P_{\text{loss}} + \nu \quad (46)$$

And finally, the mean number of message requests in the system can be written as

$$\bar{E} = \bar{E}_{EC} + \bar{E}_{CG} + \bar{E}_C. \quad (47)$$

5 Simulation and numerical results

5.1 Simulation parameters

We use the Java Modeling Tools (JMT) [35] to cross-check the results obtained by our queuing model. The JMT is an open-source discrete event simulation (DES) toolkit for simulating and evaluating the performance of computer and communication systems. JMT includes tools for workload characterization (JWAT), solution of analytical queuing networks with algorithms (JMVA), simulation of general-purpose queuing models (JSIM), bottleneck identification (JABA), and support for Markov chain models underlying queuing systems (JMCH). This simulator supports also advanced features including finite bursty processes, capacity regions, fork join servers, and load-dependent service times [36] in addition to several distributions for arrival and service process.

In our numerical examples, we assumed that the simulation environment consists of a CDC with a capacity of 20 homogenous PSs, with each PS is supposed to support at most 10 VMs. The average message arrival rate to the system is varied from 1000 to 10,000 message requests per second. The service times of message request in each EC are exponentially distributed with an average of 0.001 seconds. The maximum number of message requests in the each EC is 300. The message is either forwarded to CG for further processing in the CDC with probability $p = 0.4$. The service times of each message request in the CG are exponentially distributed with an average of 0.0001 seconds. The message request service times on each VM are independent and identically distributed exponential random variables with an average of 0.01 seconds. The maximum number of message requests in the each PS is 500. The simulation results are performed on a PC configured with a 2.40 GHz Intel Core i5 CPU, 4 GB of memory, and a 250 GB disk. For clarity, we summarize the values of all numerical parameters used in our simulation in Table 2. The choice of the values of the parameters is based on the analytical model presented in Sect. 4. We chose the different values of the parameters so that our system is stable. To validate the analytical model, we used the same assumption for analysis parameters.

Table 2 Values for system input parameters

Parameters	Description	Values
λ	Message request arrival rate	[1000 to 10,000] (req/s)
$1/\mu_1$	Mean message request EC service time	0.01 (s)
$1/\mu$	Mean message request GC service time	0.0001 (s)
$1/\mu_2$	Mean message request VM service time	0.02 (s)
C	Maximum number of message requests in each EC	300
p	The probability that the message is either forwarded to CG	0.4
K	Maximum number of message requests in each PS	500
M	Number of PSs in the data center	20
n	Number of VMs in each PS	10

5.2 Results and discussion

First, we have cross-validated our analytical results with those results obtained by JMT simulator. From the figures presented in this section, the analytical results and those obtained from the simulation are in good agreement. Hence, this validates our analytical model. The reported simulation results presented here are the average of five runs, and the average is recorded in the plots. The curves represented by lines are the results obtained from analysis, whereas the black circles represent those from simulation. We have analyzed the performance curves as function of request arrival rate using multiple ECs (44, 46, 48, and 50). Each figure presented as performance measure calculated based on the analytical model for varying messages arrival rate values and compared the simulation ones. Performance curves as those of the CPU utilization, the mean response time, the mean number of messages in the system, the system drop rate, and the mean throughput are plotted in the figures.

In Fig. 5, we plot the CPU utilization (Eq. 9) in function of messages arrival rate according to four different numbers of ECs. We have considered in our simulation 44, 46, 48, and 50 ECs. We remark that as the request arrival rate increases as the CPU utilization increases. We see that when the system has 5000 requests per second, with the usage of 50 EC, the CPU utilization tends nearly to 90% while with 44 EC the CPU utilization is 100%. Clearly, it can be concluded that for reducing the CPU utilization, it is highly recommend adding others EC in the fog computing system.

In Fig. 6, we show the impact of number of EC on the mean response time (Eq. 44) by varying the request arrival rate. It is obvious that as the request arrival rate increases, the mean response time increases. It is also observed that when the arrival rate exceeds 5000 requests per second, the response time increases rapidly from 0.2 to 2 seconds if one takes for example the case of 44 EC. This is explained in Fig. 5, in which

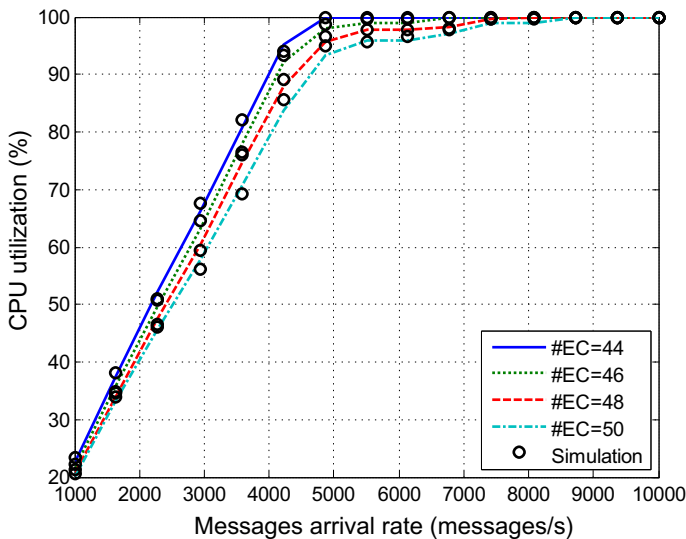


Fig. 5 Impact of number of ECs on CPU utilization

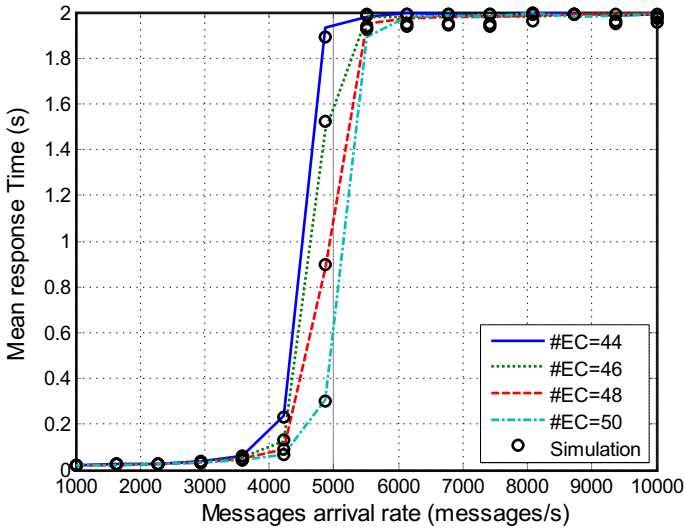


Fig. 6 Impact of number of ECs on mean response time

the CPU utilization changes to 100%, i.e., saturating the node, and obviously the response time will increase. We see that with 50 ECs, we can improve the mean response time parameter and enhance the QoS performance. For example, when there are 5000 requests per second in the system, so using 50 ECs the mean response time reaches 0.3 second while with 44 ECs the mean response time tends to 2 seconds. This demonstrates that when we increase the number of ECs in the system we give better performance, especially for the IoT system that must be prepared to attend a multiple of requests at the same time sending by different IoT devices connected in the world.

Figure 7 illustrates the impact of number of ECs on the mean number of messages in the system (Eq. 47) when the arrival rate is varied. It is clear that when we perform with higher number of ECs and the request arrival rate does not exceed 5000 requests per second, the system can process more messages which allows decreasing the number of messages in the system. In the case of the arrivals rate exceeds 5000 requests per second, the system with higher number of ECs presents a large number of messages in the system. This is explained very simply in the results obtained in Figs. 5 and 8. When the CPU utilization is equal to 100% (arrival rate exceeds 500 requests per second) and the system contains more number of ECs, the probability of dropping messages from the system decreases comparing to the system with less number of ECs.

In Fig. 8, we show the impact of the number of ECs on system drop rate (Eq. 46) while varying the request arrival rate. It is observed that when increasing the number of ECs, the drop rate of messages in the system will accordingly rise. Consequently, the system with more number of ECs yields a better performance in comparison with the system with less number of ECs. Hence, we conclude that when we processed the message requests in the fog computing layer, we subsequently improve the QoS and the SLA for IoT devices.

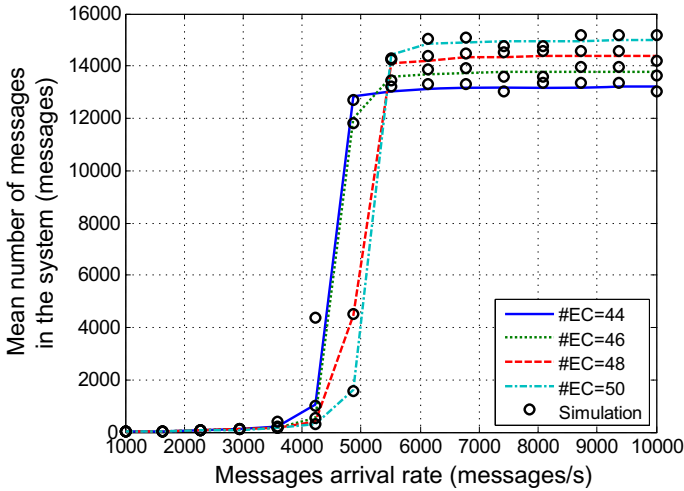


Fig. 7 Impact of number of ECs on mean number of messages in the system

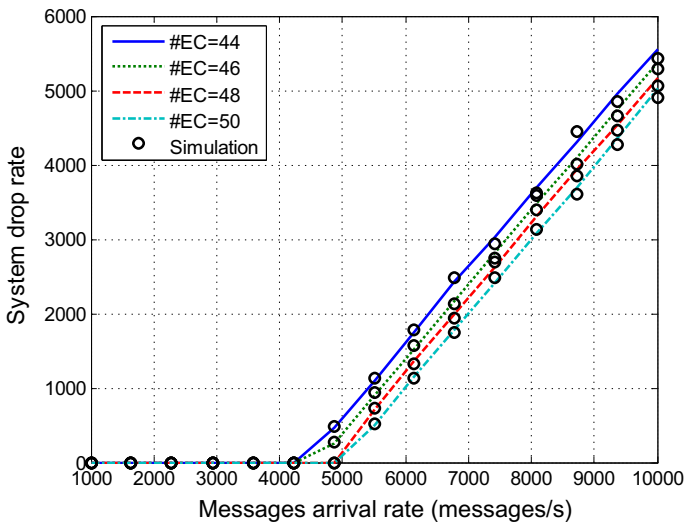


Fig. 8 Impact of number of ECs on system drop rate

Figure 9 shows the corresponding mean system throughput (Eq. 43) as a function of request arrival rate for different number of ECs. We observe that when the request arrival rate is increased, the mean system throughput increases until it levels off at the saturation point of 5000 requests per second. Beyond the saturation point, the throughput remains flat. It is noted with more EC nodes, more throughput is achieved. This is clear when eyeballing the throughput using 50 ECs and comparing it with 44 ECs at an arrival rate of 8000 requests per second. At this point, a mean system throughput delta of more than 600 requests per second has been achieved.

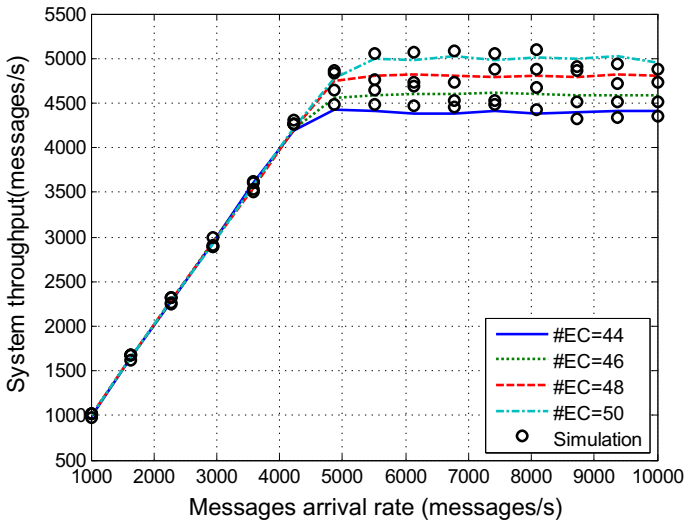


Fig. 9 Impact of number of ECs on system throughput

Figure 10 illustrates an example of how our analytical model and derived formulas can be used to achieve an efficient and dynamic scalability whereby minimal resources of EC nodes are allocated, under any workload conditions instigated by the IoT devices, to satisfy a desired SLA response time (say 30 ms). Figure 10a shows a highly fluctuating workload with a pattern of increase and decline in IoT workload represented in messages/s. Figure 10b exhibits the dynamically changing and minimum number of ECs (representing the minimal number of required EC nodes (labeled as Dynamic N) computed by our analytical model to ascertain an SLA response time below 30 ms. Figure 10c shows the corresponding mean SLA response time when using dynamic and fixed number of ECs. With a fixed number of EC (i.e., $N = 75$), it is clearly observed that the SLA response time is violated in addition to undesirable resource utilization in which over-provisioning and under-provisioning are exhibited.

6 Related work

There are many papers published on the subject of performance analysis of fog computing system. In this section, we summarize published works related to performance evaluation of fog computing environment. A novel integrated fog cloud IoT (IFCIoT) architecture is proposed in [37] with the aim of improving performance, energy efficiency, reduced latency, scalability, and better localized accuracy for IoT applications. The authors in this work also elaborate on the potential applications of IFCIoT architecture, such as intelligent transportation systems, localized weather maps and environmental monitoring, smart cities, and real-time agricultural data analytics and control. To better meet and enhance the performance, energy, and real-time requirements of applications, the authors also proposed a reconfigurable and layered fog/edge node architecture that can adapt according to the workload being run at a

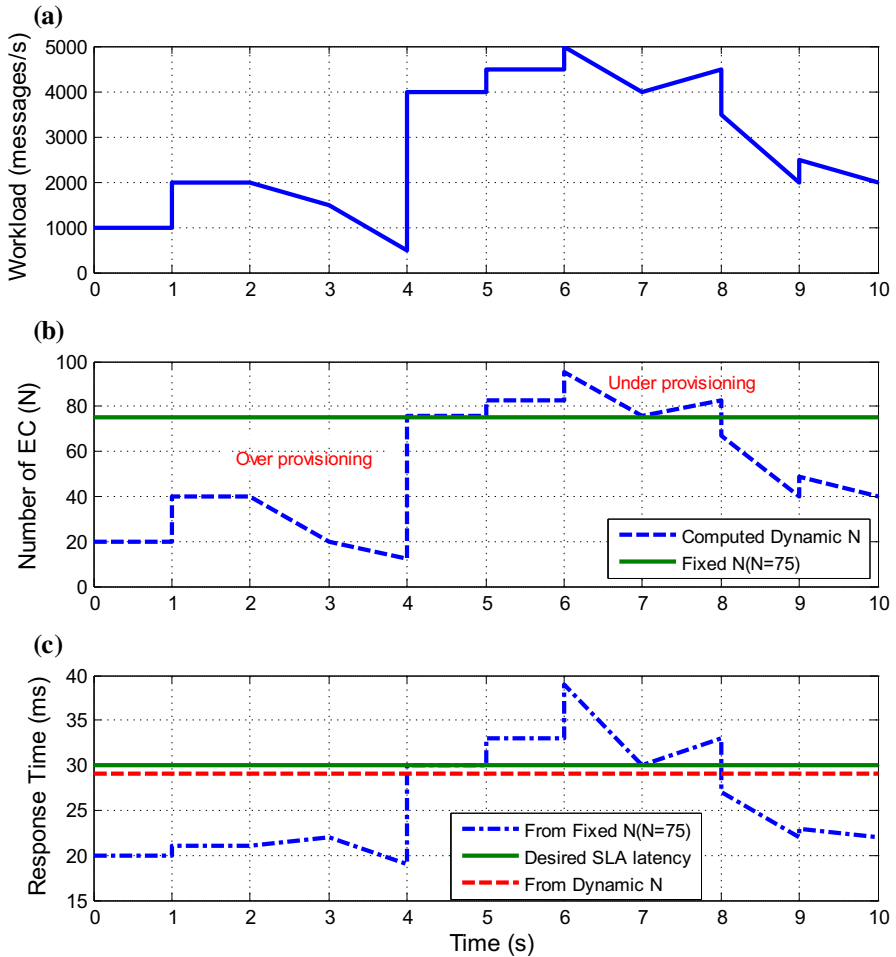


Fig. 10 Dynamic scalability with fluctuating workload

given time. Alsaffar et al. [38] introduced an architecture of IoT service delegation based on collaboration between fog and cloud computing. They provided an algorithm to allocate resources to meet the SLA and QoS constraints, as well as optimizing big data distribution in fog and cloud computing. The reported simulation results show that the proposed architecture can efficiently balance workload, improve resource allocation efficiently, optimize big data distribution, and show better performance than other existing methods.

In [39], the authors studied the interaction between the fog computing layer and the cloud layer. They provided a detailed mathematical model to describe the characteristics fog computing system in terms of power consumption, service latency, CO₂ emission, and cost. Their work shows that in the context of IoT, with high number of latency-sensitive applications fog computing outperforms cloud computing.

Urgaonkar et al. [40] modeled the fog computing system as a Markov decision problem (MDP). They provided a method using a Lyapunov optimization to minimize operational costs while providing required QoS. In [41], the authors proposed a communication architecture called smart gateway-based integration with fog computing and cloud computing. They then test their proposed architecture in terms of upload and bulk-data upload, synchronization and bulk-data synchronization, and jitter delays. Their work shows that using smart gateway-based communication with fog computing helps cloud to deliver services to applications that require low latency.

The authors in [42] designed a high-efficiency task scheduling and resources management strategy in fog computing platform designed to minimize the task completion time while providing a better QoE. Three main issues are investigated by the authors: (1) how to balance the workload on a client device and computation servers, (2) how to place task images on storage servers, and (3) how to balance the I/O interrupt requests among the storage servers. The authors in [43] used fog computing architecture for websites optimization by minimizing the HTTP requests based on the unique knowledge of the edge nodes. They improved that the increasing runtime adaptations to user's conditions can be achieved with knowledge that is specific and resides at the edge of the network. Kamiyama et al. [44] proposed a fog computing-based smart grid system. The proposed system is geographically distributed and extends the capabilities of cloud-based smart grids in terms of privacy, latency, and locality for smart grids. The distributed nature of the proposed model also provides reliability to smart grids architecture. In addition the authors presented an example scenario, which shows that the fog computing can increase the efficiency of the cloud computing-based smart grids.

The carbon emission rate of different fog nodes in terms of energy usage has been considered for resource provisioning by Do et al. in [45]. The authors developed a distributed algorithm based on the proximal algorithm and alternating direction method of multipliers (ADMM) in order to solve the large-scale optimization Internet applications problem. The reported numerical results show that the proposed algorithm converges to near optimum within fifteen iterations and is insensitive to step sizes. A practical implementation of fog computing using Raspberry Pi was attempted by Krishnan et al. in [46]. The authors proposed also a method of moving the computation from the cloud to the network by introducing an android like app store on the networking devices. The reported results show that fog computing system-based architecture has a better response time when compared to the cloud environment. Bhattacharya et al. [47] formulated a mathematical model to compare the performance of offloading from smartphone to a cloud server and user-controlled edge devices such as laptops, tablets, and routers. The obtained simulation results show that offloading to larger edge devices such as laptop can provide better performance than a cloud server. But, smaller edge devices such as tablets or routers provide slower performance than a cloud server, but can also significantly speed up application execution. They find also that offloading to user's own laptop reduces finish time of benchmark applications by 10%, compared to offloading to a commercial cloud server.

To date, and to the best of our knowledge, the literature is lacking in research aspects related to studying and modeling the performance of IoT systems leveraging the underlying infrastructure of fog and cloud computing platforms. Furthermore, the

literature is lacking to address the issue of efficient scalability of fog nodes as billions of IoT devices start getting connected at the edge, and the minimal number of fog nodes has to be allocated to meet IoT workload to meet SLA and QoS parameters. Efficient scalability is accomplished when neither provisioning nor de-provisioning of fog nodes take place. Over-provisioning results in higher cost as more fog nodes are allocated. Under-provisioning results in violation of SLA and QoS parameters.

7 Conclusion

In this paper, we have presented a mathematical queuing model to capture the dynamics and analyze the performance of fog computing systems used for IoT devices. We have derived formulas for key performance metrics such as EC CPU utilization, the system response time, the system loss rate, the system throughput, and the mean number of messages request in the system. Furthermore, we showed how our proposed model determines under any offered IoT workload the proper number of EC nodes needed to meet key SLA or QoS parameters. The analytical model has been cross-validated by simulation using the JMT simulator. We have given many numerical examples involving fog computing system with 44 ECs, 46 ECs, 48 ECs, and 50 ECs. The obtained simulation results are in line with the results of analytical model.

Acknowledgements The authors thank the anonymous reviewers for their valuable comments, which helped us to considerably improve the content, quality, and presentation of this paper.

References

1. Evans D (2011) The internet of things how the next evolution of the internet is changing everything. Technical report, CISCO IBSG
2. Botta A, De Donato W, Persico V, Pescapé A (2016) Integration of cloud computing and internet of things: a survey. *Future Gen Comput Syst* 56:684–700
3. Muhammad G, Rahman SMM, Alelaiwi A, Alamri A (2017) Smart health solution integrating IoT and cloud: a case study of voice pathology monitoring. *IEEE Commun Mag* 55(1):69–73
4. Aazam M, Khan I, Alsaffar AA, Huh EN (2014) Cloud of things: integrating internet of things and cloud computing and the issues involved. In: *Proceedings of the 11th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pp 414–419
5. Nan Y, Li W, Bao W, Delicato FC, Pires PF, Zomaya AY (2016) Cost-effective processing for delay-sensitive applications in cloud of things systems. In: *Proceedings of the 15th International Symposium on Network Computing and Applications (NCA)*, pp 162–169
6. Ab Karim MB, Ismail BI, Tat WM, Goortani EM, Setapa S, Luke JY, Ong H (2016) Extending cloud resources to the edge: possible scenarios, challenges, and experiments. In: *Proceedings of the International Conference on Cloud Computing Research and Innovations (ICCCRI)*, pp 78–85
7. Bonomi F, Milito R, Zhu J, Addepalli S (2012) Fog computing and its role in the internet of things. In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, New York, NY, USA: ACM, pp 13–16
8. Garcia Lopez P, Montresor A, Epema D, Datta A, Higashino T, Iamnitchi A, Riviere E (2015) Edge-centric computing: vision and challenges. *ACM SIGCOMM Comput Commun Rev* 45(5):37–42
9. Mehta A, Tärneberg W, Klein C, Tordsson J, Kihl M, Elmroth E (2016) How beneficial are intermediate layer data centers in mobile edge networks? In: *Proceedings of the 1st International Workshops on Foundations and Applications of Self-* Systems*, IEEE, pp 222–229
10. Dastjerdi AV, Buyya R (2016) Fog computing: helping the internet of things realize its potential. *Computer* 49(8):112–116

11. Ahmed A, Ahmed E (2016) A survey on mobile edge computing. In: Proceedings of the 10th International Conference on Intelligent Systems and Control (ISCO), pp 1–8
12. Sarkar S, Misra S (2016) Theoretical modelling of fog computing: a green computing paradigm to support IoT applications. *IET Netw* 5(2):23–29
13. Chen H, Yao DD (2013) Fundamentals of queueing networks: performance, asymptotics, and optimization, vol 46. Springer, Berlin
14. Sahner RA, Trivedi K, Puliafito A (2012) Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package. Springer, Berlin
15. Bolch G, Greiner S, de Meer H, Trivedi KS (2006) Queueing networks and Markov chains: modeling and performance evaluation with computer science applications. Wiley, New York
16. Narayan Bhat, U (2015) An introduction to queueing theory: modeling and analysis in applications. Birkhäuser, Springer, New York
17. Li W, Santos I, Delicato FC, Pires PF, Pirmez L, Wei W, Song H, Zomaya A, Khan S (2017) System modelling and performance evaluation of a three-tier cloud of things. *Future Gen Comput Syst* 70:104–125
18. Dastjerdi AV, Gupta H, Calheiros RN, Ghosh SK, Buyya R (2016) Fog computing: principles, architectures, and applications. In: Internet of things: principles and paradigms, pp 61–75, Massachusetts
19. Yuriyama M, Kushida T (2010) Sensor-cloud infrastructure-physical sensor management with virtualized sensors on cloud computing. In: Proceedings of the 13th International Conference on Network-Based Information Systems (NBIS), IEEE, pp 1–8
20. Bonomi F, Milito R, Natarajan P, Zhu J (2014) Fog computing: a platform for internet of things and analytics, big data and internet of things: a roadmap for smart environments. Springer, New York, pp 169–186
21. Misra S, Chatterjee S, Obaidat MS (2014) On theoretical modeling of sensor cloud: a paradigm shift from wireless sensor network. *IEEE Syst J PP*(99):1–10
22. Shaukat U, Ahmed E, Anwar Z, Xia F (2016) Cloudlet deployment in local wireless networks: motivation, architectures, applications, and open challenges. *J Netw Comput Appl* 62:18–40
23. Bari MF, Boutaba R, Esteves R, Granville LZ, Podlesny M, Rabbani MG, Qi Z, Zhani MF (2013) Data center network virtualization: a survey. *IEEE Commun Surv Tutor* 15(2):909–928
24. Katz RH (2009) Tech titans building boom. *IEEE Spectr* 46(2):40–54
25. Crovella M, Bestavros A (1994) Self-similarity in worldwide-web traffic: evidence and possible causes. *IEEE/ACM Trans Netw* 3(3):226–244
26. Paxson V, Floyd S (1995) Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Trans Netw* 3(3):226–244
27. Salah K, Elbadawi K, Boutaba R (2016) An analytical model for estimating cloud resources of elastic services. *J Netw Syst Manag* 24(2):285–308
28. Salah K, El Kafhali S (2017) Performance modeling and analysis of hypoexponential network servers. *Telecommun Syst*. doi:[10.1007/s11235-016-0262-3](https://doi.org/10.1007/s11235-016-0262-3)
29. Chandy KM, Sauer CH (1978) Approximate methods for analyzing queueing network models of computing systems. *J ACM Comput Surv* 10(3):281–317
30. Xiong K, Perros H (2009) Service performance and analysis in cloud computing. In: Proceedings of the 2009 IEEE Congress on Services, Los Angeles, Californian, pp 693–700
31. Burke P (2010) The output of a queueing system. *Oper Res* 4:699–704
32. Nelson R (2013) Probability, stochastic processes, and queueing theory: the mathematics of computer performance modeling. Springer, Berlin
33. El Kafhali S, Salah K (2017) Stochastic modelling and analysis of cloud computing data center. In: Proceedings of the 20th ICIN Conference Innovations in Clouds, Internet and Networks, Paris, France, March 7–9, pp 122–126
34. Dattatreya GR (2008) Performance analysis of queueing and computer networks. CRC Press, Boca Raton
35. Bertoli M, Casale G, Serazzi G (2009) JMT: performance engineering tools for system modeling. *ACM SIGMETRICS Perform Eval Rev* 36(4):10–15
36. Fishman G (2013) Discrete-event simulation: modeling, programming, and analysis. Springer, Berlin
37. Munir A, Kansakar P, Khan SU (2017) IFCIoT: integrated fog cloud IoT architectural paradigm for future internet of things. *IEEE Consum Electr Mag* (**accepted**)

38. Alsaffar AA, Pham HP, Hong CS, Huh EN, Aazam M (2016) An architecture of IoT service delegation and resource allocation based on collaboration between fog and cloud computing. *Mob Inf Syst* 2016: 1–15
39. Sarkar S, Chatterjee S, Misra S (2015) Assessment of the suitability of fog computing in the context of internet of things. *IEEE Trans Cloud Comput* PP(99):1–1. doi:[10.1109/TCC.2015.2485206](https://doi.org/10.1109/TCC.2015.2485206)
40. Uргаonkar R, Wang S, He T, Zafer M, Chan K, Leung KK (2015) Dynamic service migration and workload scheduling in edge-clouds. *Perform Eval* 91:205–228
41. Aazam M, Huh EN (2014) Fog computing and smart gateway based communication for cloud of things. In: *Proceedings of the International Conference on Future Internet of Things and Cloud, FiCloud, Barcelona, Spain 27–29 August*, pp 464–470
42. Zeng D, Gu L, Guo S, Cheng Z, Yu S (2016) Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Trans Comput* 65(12):3702–3712
43. Zhu J, Chan DS, Prabhu MS, Natarajan P, Hu H, Bonomi F (2013) Improving web sites performance using edge servers in fog computing architecture. In: *Proceedings of the 7th International Symposium on Service Oriented System Engineering (SOSE), IEEE*, pp 320–323
44. Kamiyama N, Nakano Y, Shiimoto K, Hasegawa G, Murata M, Miyahara H (2016) Priority control based on website categories in edge computing. In: *Proceedings of the Conference on Computer Communications Workshops (INFOCOM WKSHPS), IEEE*, pp 776–781
45. Do CT, Tran NH, Pham C, Alam MGR, Son JH, Hong CS (2015) A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing. In: *Proceedings of the International Conference on Information Networking (ICOIN), IEEE, Cambodia*, pp 324–329
46. Krishnan YN, Bhagwat CN, Utpat AP (2015) Fog computing—network based cloud computing. In: *Proceedings of the 2nd International Conference on Electronics and Communication Systems (ICECS), IEEE, Coimbatore, India*, pp 250–251
47. Bhattacharya A, De P (2016) Computation offloading from mobile devices: Can edge devices perform better than the cloud?. In: *Proceedings of the Third International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing, ACM*, pp 1–6