Autonomic Resource Management for Fog Computing

Uma Tadakamalla, *Member, IEEE* and Daniel A. Menascé, *Fellow, IEEE*

Abstract—Fog computing is a distributed computing paradigm that extends cloud computing capabilities to the edge of the network and aims at reducing high latency and network congestion, which are characteristics of cloud computing. This recent paradigm enables portions of a transaction to be executed at a fog server and other portions at the cloud. Fog servers are generally not as robust as cloud servers; at peak loads, the data that cannot be processed by fog servers is processed by cloud servers. The data that need to be processed by the cloud is sent over a Wide Area Network (WAN). Therefore, only a fraction of the total data needs to travel through the WAN, as compared with a pure cloud computing paradigm. Additionally, the fog/cloud computing paradigm reduces the cloud processing load when compared with the pure cloud computing model. This article presents a multiclass closed-form analytic queuing network model that is used by an autonomic controller to dynamically change the fraction of processing between edge and cloud servers in order to maximize a utility function of response time and cost. The model was validated using both synthetic and real IoT traces. A detailed design of the autonomic controller is presented and a series of experiments compare the efficacy and efficiency of the controller versus a brute force optimal controller and versus an uncontrolled system using synthetic and real traces. The results show that the controller is able to maintain a high utility in the presence of wide variations of request arrival rates.

Index Terms—Fog computing, cloud computing, edge computing, analytic queuing models, resource management, autonomic control

1 Introduction

Fog computing is a relatively new distributed computing paradigm that extends cloud computing capabilities to the edge of the network with the goal of reducing network latency and network congestion [1], [2]. Fog resources/servers reside closer to end-user devices and act as an intermediate layer between cloud datacenters and end-user devices. These resources provide compute, storage, and networking services between these devices and traditional clouds. This paradigm is more suitable for applications with low latency requirements such as Internet of Things (IoT) services. Example services include connected cars (self-driven or semi-autonomous), health-care systems that monitor patients remotely, smart city fog servers located on traffic intersections, on-ramp freeway vehicle admission control devices, reconnaissance and traffic control drones.

Fog computing enables portions of a transaction to be executed at a fog server and other portions at the cloud. Because fog servers are generally not as robust as cloud servers, the data that cannot be processed by fog servers at peak loads is processed by cloud servers [3]. The data that need to be processed by the cloud is sent over a Wide Area Network (WAN). Therefore, only a fraction of the total data needs to travel through the WAN, as compared with a pure cloud computing paradigm. Thus, fog computing reduces the cloud processing load when compared with the pure cloud computing model. Therefore, fog computing can perform efficiently in terms of service latency, power consumption, network traffic and operational expenses and improve response times [1].

By their very nature, fog servers are resource-constrained. For that reason, we present in our work an efficient and lightweight closed-form autonomic controller that can be used in applications that need a mechanism for making online decisions on where to run a request (at the edge or at the cloud).

Problem Motivation. A key resource management issue when analyzing and optimizing fog computing systems is the determination of the impact of the fraction f of data processing executed at the cloud versus at fog servers. Several tradeoffs need to be considered. The processing capacity of fog servers is typically smaller than that of cloud servers because the latter have ample capacity, which can be increased in an elastic way. As f increases, more data has to be sent and received from the cloud. Given that fog servers communicate with the cloud over the WAN, more round-trip times and transmission times add up to the cloud processing time. On the other hand, fog servers may not have enough capacity to handle requests from sensors or other IoT devices and may become a bottleneck.

For example, Fig. 1 illustrates this tradeoff by indicating the variation of the average response time of requests as a function of f for two different times of a day: 6am and 3pm. The figure shows that there is an optimal value of f that minimizes the average response time for different values of the average arrival rate of requests. The average arrival rates used in the figure are 8.5 req/sec and 7.5 req/sec, and come from a sample of CityPulse traffic data [4]. The figure

• *The authors are with the Department of Computer Science, George Mason University, Fairfax, VA 22030 USA.*
E-mail: utadakam@masonlive.gmu.edu, menasce@gmu.edu.

Manuscript received 15 Sept. 2020; revised 21 Jan. 2021; accepted 3 Mar. 2021. Date of publication 9 Mar. 2021; date of current version 6 Dec. 2022.
(Corresponding author: Daniel A. Menascé.)
Recommended for acceptance by J. Catal?o.
Digital Object Identifier no. 10.1109/TCC.2021.3064629

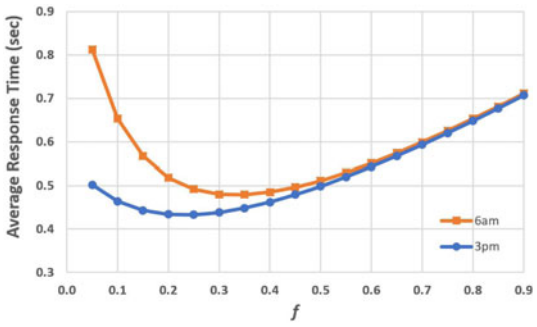

Fig. 1. Average response time (in sec) versus fraction of cloud processing (f) for two different average arrival rates of 8.5 req/sec and 7.5 req/sec at 6am and 3pm of a day, respectively.

indicates for both curves that for lower values of f, the average response time decreases as more processing is shifted to the cloud. However, for higher values of f, the response time starts to increase due to excessive use of cloud services and data transmission over the WAN. Therefore, the optimal values of f that results in lowest response times are approximately 0.35 and 0.25 for the 6am and 3pm traffic data, respectively, as shown in the figure. So, the optimal fraction f_{opt}, varies with the arrival rate. Thus, it is necessary to dynamically adjust f in an autonomic manner to maintain an optimal or near-optimal system utility, which is typically a function of average response times and costs.

For a given f, simulation can be used to estimate the average response time of requests. In this very simple example with just one class of requests, eighteen simulation runs are needed in order to compute f_{opt}, i.e., one simulation for each $f \in \{0.05, 0.10, ..0.90\}$. In contrast, a general case consists of multiple classes of requests, R, with inputs as average arrival rates of requests, λ_r for $r \in \{1, 2, ..R\}$. The corresponding number of simulation runs (n) required to compute optimal fractions, $f_{opt,r}$ for $r \in \{1, 2, ..R\}$, grows exponentially with R, i.e., $n = \mathcal{O}(constant^R)$, which is $\approx 18^R$ for the example above with multiple classes of requests using a brute force method. With very efficient, mathematical, computational optimization, and/or near optimal greedy techniques, the computational complexity for n can be reduced from exponential to polynomial; but, that still may be a very large number for many real-world cases. Since, conducting even a single simulation run is generally very expensive and time consuming, we need an alternative solution in the form of analytic models. In other words, the advantages of an analytic model as opposed to a simulation-based model is that the computation using analytic equations is orders of magnitude faster than running simulations and is generally inexpensive. Additionally, analytic equations allow us to perform sensitivity analyses as a function of the model parameters. To the best of our knowledge, there are no such analytic models for fog computing; therefore there is a need for research in that area.

Contributions. This paper shows that it is possible to use analytic queuing networks to build an analytic model for fog computing to estimate performance and cost metrics given the characteristics of workloads and the fraction of data processing at the cloud. It is further possible to use the fog computing model in the design of an autonomic controller that dynamically changes the fraction of the workload that is processed at the cloud. The controller attempts to optimize or find a near optimal value of a system utility, which typically is a function of performance and cost metrics. Thus, the key contributions of this paper are:

- Development, implementation, and validation of an analytic model, called *FogQN*, for fog computing to estimate performance and cost metrics. The analytic model is based on open multi-class Queuing Networks (QN). In addition, we developed a corresponding tool, called *FogQN*, which is publicly available at [5]. The model was validated with the JMT simulation tool using both distribution-based arrival rates and inputs from several publicly available real IoT traces. We published a preliminary version of *FogQN* in [6]. Here we extend the model as follows: (i) we added security and backup options to fog requests and modified the model equations accordingly (see Section 2), (ii) We enhanced the model validation section by using two additional IoT traces (Chicago Taxi Dataset and Beijing Taxi Dataset) (see Section 3.2) (iii) We analyzed the effect of security and backup options on the model (see Section 4).
- Design, implementation, and assessment of an autonomic controller, called *FogQN-AC*, for fog computing. The autonomic controller dynamically changes the fraction of data processing performed at the cloud as the workload varies over time. The controller seeks to optimize a system utility function of average response times and costs. We also developed a corresponding tool, called *FogQN-AC*. The controller is assessed against a brute-force optimal solution that performs an exhaustive search to find an optimal fraction. The results show that the controller is very efficient in finding an optimal or a near optimal solution. Additionally, we conducted experiments to assess the controller against the no controller case, i.e., static f, using both synthetic traces and real traces. The static experiments start with the optimal value of f. The experiments show that the controller is able to maintain a high utility in the presence of wide variations of request arrival rates. A preliminary version of *FogQN-AC* appeared in [7]. This paper extends the controller and its evaluation as follows: (i) The specification of the optimization model and controller algorithms were adapted to include security and backup options (see Section 5), (ii) We compared the computational effort and efficiency of *FogQN-AC* versus that of a mathematical solver (see Section 6.6. (iii) We assessed the effect of backup and security options on the utility improvement (see Section 6.7). (iv) For all comparisons between *FogQN-AC* and an uncontrolled system we started the uncontrolled system at the optimal state (see Section 6).

Organization. Section 2 presents the analytic queuing model *FogQN* including its assumptions, notation and parameters, computation of service demands, response time and cost equations. The model is validated in Section 3 using Poisson arrivals and three different IoT traces.

Section 4 discusses the effect of various security and backup option combinations on response time and cost. The next section presents a detailed design of *FogQN-AC*, an autonomic controller for fog computing environments that uses *FogQN*. Section 6 discusses the results of several experiments aimed at validating *FogQN-AC*. These experiments include validations with synthetic and real traces from IoT applications and compare the effectiveness of *FogQN-AC* versus a brute-force optimal controller and with a mathematical non-linear solver. Section 7 discusses related work in the areas of fog computing, the application of Queuing Networks to fog and cloud computing, and autonomic controllers. The last section presents some concluding remarks.

2 *FogQN*: An Analytic Model for Fog/Cloud Computing

FogQN is an analytic model and corresponding tool based on multi-class open Queuing Networks (QN) [8] to analyze the performance and cost of requests processed by a fog computing environment. This model can be used in the design of an autonomic controller that dynamically determines the best partition between fog and cloud processing. We implemented the model and corresponding tool, also known as *FogQN*, which is publicly available at [5].

There can be two types of system models for fog/cloud computing based on where transactions are processed. In the first model, a transaction is processed entirely either on a fog or on a cloud server. In the second model, a transaction can be split between fog and cloud servers. The *FogQN* analytic model presented here applies for both types of system models.

2.1 The QN Model of *FogQN*

Multi-class open QN models have been well studied (see e.g., [8]). For completeness sake, we present here a brief overview of this model. A QN is a network of K *queues* where a queue consists of a single waiting line and one or more servers. For instance, a queue can represent a server, a network, a processor, or an I/O device. In open QNs, arriving requests visit various queues, some more than once, until they complete processing and leave the system. Requests are grouped into clusters, aka *classes* in the QN literature, of requests that have similar demands on the various queues; the demands could be arrival rate, service time, backup and security characteristics. The average total service time spent by a class r ($r = 1, \ldots, R$) request at a queue k ($k = 1, \ldots, K$) is called the *service demand* and is denoted by $D_{k,r}$.

Fig. 2 shows the *FogQN* model, an open QN that models the Fog/Cloud environment with n fog servers and m cloud servers. The QN shows an arrival stream of requests that can be generated for example from IoT devices. These requests go through a local area network (LAN) and reach one of the n fog servers. Each fog server (FS) consists of various queues including one or more CPUs and disk queues. Requests may cycle through the devices of a FS and may complete processing after using only FS devices or may require additional processing by the cloud. In the latter case, such cloud requests have to go through the WAN and be processed by the devices (CPU and disks) of a cloud service provided by one of the m cloud servers. Readers familiar with QN theory will recognize that any number of queues can be easily modeled by an open QN (see [8]).

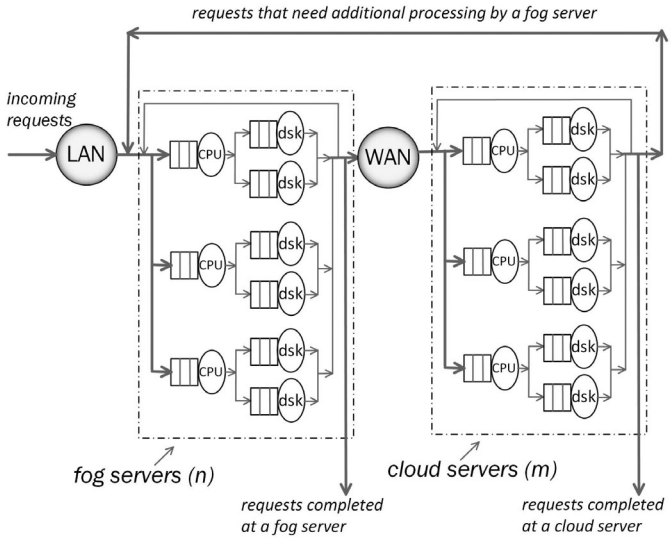

Fig. 2. Queuing model for fog computing.

Thus, the queues considered by *FogQN* are: (a) LAN: represents the time (i.e., latency plus transmission time) spent by a request in the local area network that connects sources of requests (e.g., IoT devices) to fog servers; (b) Fog Server: represents the waiting and processing time at a fog server. Note that a request may have to visit a fog server more than once and wait in line each time; (c) WAN: represents the time (i.e., latency plus transmission time) spent at the Wide Area Network to send requests from a fog server to the cloud and receive replies; and (d) Cloud Server: represents the time spent by a request waiting and being processed by a cloud server. As the figure illustrates, a request may perform several cycles during which it visits a fog server, the WAN, and a cloud server, until it completes. A request may complete at the fog server or at the cloud server.

2.2 Types of Requests

Requests can be classified *by users* as secure requests and non-secure requests based on their security characteristics. Secure requests can only be processed entirely on fog servers, and cannot be sent to the cloud via the WAN. Non-secure requests can be processed on fog/cloud servers and can be transmitted on the WAN to the cloud.

Similarly, requests can be classified *by users* based on their backup requirements. Requests that require backup need to transmit the data that was processed on the fog servers to the cloud via the WAN. So, the total cost of processing backup requests includes additional transmission cost to send the data to the cloud and the data storage cost at the cloud.

2.3 Assumptions

- A1: Fog and cloud servers are modeled as load independent queues, i.e., they have a finite capacity of work and their processing rate does not depend on the queue length.
- A2: The LAN and WAN are modeled as delay queues, i.e., they have ample processing capacity

(aka delay queue) and have no queuing of requests; thus, the residence time is simply the service demand $D_{k,r}$. Also, the processing rate does not depend on the queue size.
- A3: The traffic at the fog layer is uniformly balanced among all n fog servers
- A4: The traffic at the cloud is uniformly balanced among all m cloud servers.
- A5: Requests of the same class have similar arrival rates, service demands, amount of data to be processed, backup and security characteristics.
- A6: Requests that require backup transmit data processed at fog servers to the cloud via the WAN. However, the response is sent to the requesting client/device immediately after processing before backup data is sent to the cloud. Thus, the response time of a request does not depend on whether the request requires backup or not; however, the processing cost of requests that need backup is higher than those that do not need backup.
- A7: Secure requests have to be processed entirely on fog servers and cannot be sent to the cloud via the WAN. Thus, secure requests cannot use cloud backup services.

2.4 Notation and Parameters

The notation and parameters used in our model are divided into application-level and infrastructure-level parameters.

2.4.1 Application-Level Parameters

- R: number of classes of requests
- λ_r: average arrival rate of class r requests (req/sec)
- L_r: amount of data to be processed by class r requests at either a fog server or cloud server (MB)
- f_r: fraction of the data L_r processed at the cloud. Let $\vec{f} = (f_1, \ldots, f_R)$
- b_r: backup service indicator for class r requests; $b_r = 1$ for backup (b) requests and $b_r = 0$ for non-backup (nb) requests. See assumption A6. Let $\vec{b} = (b_1, \ldots, b_R)$
- s_r: security level indicator for class r requests; $s_r = 1$ for non-secure (ns) requests and $s_r = 0$ for secure (s) requests. According to assumption A7, $s_r = 0 \Rightarrow f_r = 0$ and $b_r = 0$. Let $\vec{s} = (s_1, \ldots, s_R)$
- d_r: average size of a data packet sent from the cloud to the fog server for class r requests (MB)
- $I_r^{\max}$: maximum number of cloud service invocations per class r request.
- I_r: average number of cloud service invocations per class r request. We assume that $I_r = f_r \times I_r^{\max}$.
- γ_r^{fc}: total amount of data transmitted by class r requests from the fog server to the cloud (MB); we assume that $\gamma_r^{fc} = s_r \times f_r \times L_r$. This does not include the data that was processed at the fog servers transmitted to the cloud for storage.
- γ_r^{cf}: total amount of data transmitted by class r requests from the cloud to the fog server (MB); we assume that $\gamma_r^{cf} = s_r \times I_r \times d_r = s_r \times f_r \times I_r^{\max} \times d_r$.

2.4.2 Infrastructure-Level Parameters

- n: number of fog servers
- m: number of cloud servers
- B_{LAN}: bandwidth of the LAN connecting an IOT device and the fog server (MB/sec)
- B_{WAN}: bandwidth of the WAN connection between a fog server and the cloud (MB/sec)
- LAT_{LAN}: round-trip latency of the LAN connecting an edge device to the fog server (sec)
- LAT_{WAN}: round-trip latency of the WAN connection between a fog server and the cloud (sec)
- δ: scaling factor used to multiply any portion of a request's processing time that has to be processed at the cloud
- C_{LAN}: cost of sending data through the LAN connecting edge devices to a fog server ($/MB)
- C_{WAN}: cost of sending data through the WAN connecting a fog server to the cloud ($/MB)
- C_f: Processing cost per unit time at the fog server ($/sec)
- C_c: Processing cost per unit time at the cloud ($/sec)
- C_s: Storage cost per MB at the cloud ($/MB)

2.4.3 Service Demand Parameters

These parameters are computed as a function of the application level and infrastructure parameters.

- $D_{f,r}$: average service demand of class r requests at a fog server (sec)
- $D_{c,r}$: average service demand of class r requests at a cloud server (sec)
- $D_{\text{LAN},r}$: average service demand of class r requests at the LAN (sec)
- $D_{\text{WAN},r}$: average service demand of class r requests on the WAN (sec)

2.5 Model Outputs

The outputs of FogQN are denoted as:

- $R'_{\text{LAN},r}$: residence time of class r requests at the LAN (sec)
- $R'_{f,r}$: residence time of class r requests at a fog server (sec)
- $R'_{\text{WAN},r}$: residence time of class r requests at the WAN (sec)
- $R'_{c,r}$: residence time of class r requests at the cloud (sec)
- R_r: average response time of class r requests (sec)
- C_r: average processing cost of class r requests ($/1000 req). This does not include infrastructure costs.

2.6 Computation of Service Demands

The service demand per class r request at the LAN is the sum of the LAN's latency and the transmission time. Thus

$$D_{\text{LAN},r} = \frac{L_r}{B_{\text{LAN}}} + LAT_{\text{LAN}}. \quad (1)$$

We assume that the average total processing time of class r requests (sec) at a fog server or at the cloud is equal to the sum of two terms: a constant term K_1 (in sec) and another proportional to the amount of data processed. The proportionality factor is denoted as K_2 (in sec/MB). Thus, the

service demand at the fog server per class r requests can be written as

$$D_{f,r} = \frac{1}{n}\Big(K_1 + K_2(1 - s_r\ f_r)L_r\Big). \tag{2}$$

The term $1/n$ in Eq. (2) is due to Assumption A3. The second term in Eq. (2) is the processing time at a fog server that is proportional to the amount of data $(1 - s_r f_r)L_r$ processed at the fog server.

The service demand at the WAN for class r requests is the sum of the latency $I_r \times LAT_{\text{WAN}}$ and the transmission time between the fog server and the cloud and vice-versa. Each interaction with the cloud incurs in a latency. Hence, because we assumed that $\gamma_r^{fc} = s_r f_r L_r$ and that $\gamma_r^{cf} = s_r f_r I_r^{\max} d_r$, we get

$$\begin{aligned} D_{\text{WAN},r} &= s_r\ f_r\ I_r^{\max}\ LAT_{\text{WAN}} + \frac{\gamma_r^{fc} + \gamma^{cf}}{B_{\text{WAN}}} \\ &= s_r\ f_r\Big[I_r^{\max} LAT_{\text{WAN}} + \frac{L_r + I_r^{\max} d_r}{B_{\text{WAN}}}\Big]. \end{aligned} \tag{3}$$

Finally, the service demand for class r requests at the cloud can be written as

$$D_{c,r} = \frac{1}{m}\ s_r\ \delta\Big(K_1 + K_2 f_r L_r\Big), \tag{4}$$

because δ is the scaling factor for processing time at the cloud, m is the number of cloud servers, $s_r f_r L_r$ is the portion of the data processed at the cloud. The factor $1/m$ is due to Assumption A4.

2.7 Response Time Calculation

The residence time equations for the LAN, fog server, WAN, and the cloud are computed based on [8] and on the service demands computed above

$$R'_{\text{LAN},r} = D_{\text{LAN},r}; \quad R'_{\text{WAN},r} = D_{\text{WAN},r} \tag{5}$$

$$R'_{f,r} = \frac{D_{f,r}}{1 - \sum_{s=1}^{R} \lambda_s D_{f,s}} \tag{6}$$

$$R'_{c,r} = \frac{D_{c,r}}{1 - \sum_{s=1}^{R} \lambda_s D_{c,s}}. \tag{7}$$

Finally, the response time of class r requests is given by the sum of the residence times at the LAN, fog server, WAN, and the cloud. So

$$R_r = R'_{\text{LAN},r} + R'_{f,r} + R'_{\text{WAN},r} + R'_{c,r}. \tag{8}$$

Because R_r is a function of the service demands of all classes at the fog server and at the cloud (see Eqs. (6) and (7)) and these service demands depend on all values of f_r for $r = 1, \ldots, R$, we will use the notation $R_r(\vec{f})$ to make that dependence explicit.

2.8 Cost Calculation

The total cost C_r of processing a class r request is the sum of the costs of transmitting data through the LAN and WAN, plus the processing costs at the fog server and the cloud, plus backup costs for non-secured backup requests (s_r = ns, b_r = b), which is the sum of the cost of transmitting the data that was processed at the fog server to the cloud through the WAN and the cost of storing non-secure data at the cloud. The transmission cost at the LAN is obtained by multiplying the total amount of data transmitted over the LAN by the cost per MB. Thus, this cost is $L_r \times C_{\text{LAN}}$. The cost of data transmission over the WAN is $(\gamma_r^{fc} + \gamma_r^{cf}) \times C_{\text{WAN}}$ following the same approach as for the LAN. The processing cost at the fog server is obtained by multiplying the residence time at the fog server by the cost per unit time C_f at the fog server. Thus, this cost is $R'_{f,r} \times C_f$. Using a similar reasoning, the processing cost at the cloud is $R'_{c,r} \times C_c$. The transmission cost of non-secure requests for backup is the cost of sending the data that was processed at the fog server to the cloud. This is obtained by multiplying the amount of data processed at the fog by C_{WAN}. Thus, this cost is $b_r s_r L_r(1 - f_r) \times C_{\text{WAN}}$. The storage cost for backing up data of non-secure requests at the cloud server is obtained by multiplying the amount of data by the storage cost per MB. Thus, this cost is $b_r s_r L_r C_s$.

Therefore, the average cost of class r requests is

$$\begin{aligned} C_r = {} & L_r \times C_{\text{LAN}} + (\gamma_r^{fc} + \gamma_r^{cf}) \times C_{\text{WAN}} \\ & + R'_{f,r} \times C_f + R'_{c,r} \times C_c \\ & + b_r s_r [L_r \times (1 - f_r) \times C_{WAN} + L_r \times C_s]. \end{aligned} \tag{9}$$

As above, we denote C_r as $C_r(\vec{f})$.

3 *FogQN* Model Validation

We validated the equations of the *FogQN* model presented in Section 2 with two types of simulations using Java Modeling Tools (JMT) [9]. JMT is an integrated framework of Java tools for performance evaluation of computer systems. The JMT suite consists of six tools for: performance evaluation, capacity planning, workload characterization, and modeling of computer and communication systems. The algorithms in this suite allow for simulation analysis of queuing network models. We validated *FogQN* using two of the tools from the JMT suite: JSIM*wiz* and JSIM*graph* [9].

In the first type of validation we used Poisson arrivals and in the other we used interarrival time distributions obtained from several publicly available real trace datasets. In addition, we have also conducted several experiments to study the effect of backup and security options of the requests on the average response times and cost.

3.1 Validation With Poisson Arrivals

In this section, we validated the *FogQN* model using synthetic data. The parameters used for the validation of a two-class model are shown in Table 1 and the validation results are shown in Table 2. The LAN and WAN latency and bandwidth values come from typical values found in [10]. The arrival process is assumed to be Poisson, i.e., the interarrival time of requests is assumed to be exponentially distributed. Note from the top of Table 2 that we kept λ_2 constant at 3.0 req/sec throughout the experiments reported in that table and increased λ_1 from 1.0 req/sec to 3.0 req/sec. The

TABLE 1
Model Parameters Utilized and Computed Service Demands

[A] Application-level Parameters		
R	2 classes	
$\vec{\lambda}$	[varies, 3 req/sec]	req/sec
$\vec{L}$	[4.0, 7.0]	MB/req
$\vec{f}$	[0.30, 0.40]	
$\vec{d}$	[0.05120, 0.1024]	MB/req
$\vec{I}^{max}$	[1, 1]	
K_1	0.015	sec/req
K_2	0.032	sec/MB
$\vec{b}$	[nb, nb]	
$\vec{s}$	[ns, ns]	
[B] Infrastructure-level Parameters		
B_{LAN}	1250	MB/sec
B_{WAN}	5.00	MB/sec
LAT_{LAN}	0.0008	sec/req
LAT_{WAN}	0.04	sec/req
n	1	
m	1	
C_{LAN}	0.00001	($/MB)
C_f	0.00115741	($/sec)
C_{WAN}	0.0001	($/MB)
C_c	0.00115741	($/sec)
$C_{storage}$	0.00002	($/MB)
δ	0.4	
[C] Computed values		
$\vec{I}_r$	$= f_r \times I_r^{max}$ = [0.30, 0.40]	
$\vec{\gamma}^{cf}$	$= s_r \times I_r \times d_r$ = [0.01536, 0.04096]	MB/req
$\vec{\gamma}^{fc}$	$= s_r \times f_r \times L_r$ = [1.200, 2.800]	MB/req
[D] Computed Service Demands		
$\vec{D}_{LAN}$	[0.0040, 0.0064]	sec/req
$\vec{D}_f$	[0.1046, 0.1494]	sec/req
$\vec{D}_{WAN}$	[0.2551, 0.5842]	sec/req
$\vec{D}_c$	[0.0214, 0.0418]	sec/req

absolute percent error, ϵ, between the analytical and simulation models is computed as

$$\epsilon_r = |\,(100 \times (simulation - analytical)/simulation)\,|. \tag{10}$$

As Table 2 clearly shows, the absolute percent error is very small in most cases and ranges from 1.3 to 11.9 percent but stays below 10 percent in the vast majority of cases. The absolute average errors are approximately 10.0 and 2.8 percent for classes 1 and 2, respectively. The overall average absolute error is approximately 6.4 percent, which is very low. Therefore, we consider the analytic expressions used by *FogQN* as validated using Poisson arrivals.

3.2 Validation With Real Traces

We used three existing IoT traces to explore the robustness of *FogQN* for interarrival time distributions obtained from real IoT applications: Seattle Bike Counter (SEATTLE) [11], Chicago Taxi dataset (CHICAGO) [12], and Beijing Taxi dataset (BEIJING) [13].

The City of Seattle Open Data portal [11] releases several datasets that are generated by IoT devices in Seattle. We used the "NW 58th St Greenway at 22nd Ave NW Bike Counter" dataset, which we call SEATTLE, for our validation. This dataset is generated by two IoT devices, called tube sensors, which are installed across the street, and count the number of bicyclists and sends the information to an eco-counter server. The dataset has records that contain the number of bikes that traverse the east and west directions of the street at every hour, which will be referred as classes 1 and 2, respectively, heretofore.

TABLE 2
Mean and 95 percent Confidence Intervals of Average Response Times Using *FogQN* and Simulation (JMT)

λ_1 = varies; λ_2 = 3.0 req/sec						
	R_1 (sec)			R_2 (sec)		
λ_1	FogQN	Simulation	$\epsilon(\%)$	FogQN	Simulation	$\epsilon(\%)$
1.0	0.518	0.559 ± 0.017	7.4	0.974	0.955 ± 0.027	2.0
1.2	0.530	0.580 ± 0.012	8.7	0.990	0.971 ± 0.018	2.0
1.4	0.542	0.595 ± 0.013	8.8	1.009	0.996 ± 0.029	1.3
1.6	0.557	0.613 ± 0.017	9.2	1.029	1.005 ± 0.015	2.4
1.8	0.572	0.634 ± 0.011	9.8	1.052	1.009 ± 0.022	4.2
2.0	0.590	0.652 ± 0.017	9.5	1.077	1.054 ± 0.027	2.2
2.2	0.610	0.686 ± 0.020	11.0	1.106	1.087 ± 0.025	1.7
2.4	0.633	0.711 ± 0.021	11.0	1.138	1.103 ± 0.020	3.2
2.6	0.659	0.748 ± 0.020	11.9	1.176	1.134 ± 0.029	3.7
2.8	0.689	0.781 ± 0.022	11.7	1.219	1.183 ± 0.032	3.0
3.0	0.725	0.819 ± 0.021	11.5	1.270	1.207 ± 0.034	5.2
Avg			10.0			2.8

The CHICAGO dataset is provided by the City of Chicago's open data portal [12] and contains information on taxi trips in Chicago. We computed the interarrival times of taxi trip records for February 22 and 23, which we called classes 1 and 2, respectively.

For the BEIJING dataset we used the Microsoft T-Drive Trajectory dataset, obtained from the Microsoft taxi trips dataset [13]. This dataset contains the GPS trajectories of 10,357 taxis (one file per taxi) during the period of February 2–8, 2008 within Beijing. We used February 3 and 4, 2008 taxi interarrival times for classes 1 and 2, respectively.

Table 3 shows a summary of the results of the validation of *FogQN*'s predicted response time values for classes 1 and 2 for the three traces against simulation results using the Java Modeling Tool (JMT) driven by the traces. The table

TABLE 3
Mean and 95 percent CI of Response Times Using *FogQN* and Simulation (JMT) for SEATTLE, CHICAGO, and BEIJING. $\vec{f}$ = [0.3, 0.5]

SEATTLE					
λ_1 (East) = 12.192 req/sec; λ_2 (West) = 9.205 req/sec;					
R_1 (sec)			R_2 (sec)		
FogQN	Simulation	$\epsilon(\%)$	*FogQN*	Simulation	$\epsilon(\%)$
0.234	0.259 ± 0.007	9.7	0.268	0.299 ± 0.007	10.4
CHICAGO					
λ_1 (Feb 22, 2015) = 0.1879 req/sec; λ_2 (Feb 23, 2015) = 0.2812 req/sec;					
R_1 (sec)			R_2 (sec)		
FogQN	Simulation	$\epsilon(\%)$	*FogQN*	Simulation	$\epsilon(\%)$
0.392	0.385 ± 0.007	1.8	0.923	0.928 ± 0.020	0.5
BEIJING					
λ_1 (Feb 3, 2008) = 0.6993 req/sec; λ_2 (Feb 4, 2008) = 0.7680 req/sec;					
R_1 (sec)			R_2 (sec)		
FogQN	Simulation	$\epsilon(\%)$	*FogQN*	Simulation	$\epsilon(\%)$
0.408	0.420 ± 0.012	2.9	0.944	0.991 ± 0.028	4.7

TABLE 4
Comparison of Average Response Times for Various Security ($\vec{s}$) and Backup ($\vec{b}$) Combinations

$[\lambda_1, \lambda_2] = [1.0, 3.0]$ req/sec; R_1, R_2 in sec								
	Backup, $\vec{b}$							
Security, $\vec{s}$	[nb, nb]		[nb, b]		[b, nb]		[b, b]	
	R_1	R_2	R_1	R_2	R_1	R_2	R_1	R_2
[ns, ns]	0.518	0.974	0.518	0.974	0.518	0.974	0.518	0.974
[ns, s]	0.867	1.346	Invalid option		0.867	1.346	Invalid option	
[s, ns]	0.354	1.004	0.354	1.004	Invalid option		Invalid option	
[s, s]	1.025	1.714	Invalid option		Invalid option		Invalid option	

TABLE 5
Comparison of Average Processing Costs for Various Security ($\vec{s}$) and Backup ($\vec{b}$) Combinations

$[\lambda_1, \lambda_2]$ = [1.0, 3.0] req/sec; Note: C in \$/1000 req								
	Backup, $\vec{b}$							
Security, $\vec{s}$	[nb, nb]		[nb, b]		[b, nb]		[b, b]	
	C_1	C_2	C_1	C_2	C_1	C_2	C_1	C_2
[ns, ns]	0.461	0.798	0.461	1.358	0.821	0.798	0.821	0.798
[ns, s]	0.865	1.621	Invalid option		1.225	1.621	Invalid option	
[s, ns]	0.445	0.832	0.445	1.392	Invalid option		Invalid option	
[s, s]	1.222	2.046	Invalid option		Invalid option		Invalid option	

also shows the absolute percent error, ϵ, for all cases and the 95 percent confidence intervals for the simulation results. The table indicates that the errors are below 10 percent in all cases except for one in which the error is 10.4 percent. Therefore, we consider the model as validated for a variety of real-life trace inputs.

4 Effect of Backup and Security Options on Response Times and Costs

We studied the effect of backup and security options on response times and cost of processing requests. The input parameters are the same as those in Tables 1 [A] and 1 [B], except for the vectors $\vec{b}$ and $\vec{s}$ which vary. The arrival rates are set at $\vec{\lambda} = [1, 3]$ req/sec. The computed response times and costs of processing requests for various backup and security options using *FogQN* are shown in Tables 4 and 5, respectively.

4.1 Response Time Effect

Table 4 shows that the response times for secure requests are higher than for non-secure requests. For example, R_2 is 0.974 sec (row 1) for non-secure non-backup requests and 1.346 sec (row 2) for secure non-backup requests. Similarly, R_1 for non-secure non-backup requests is 0.867 sec (row 2) and 1.025 sec (row 4) for secure non-backup requests. The increase in response time is due to the fact that secure requests can only be processed on fog servers, which are not as powerful as cloud servers. This results in increased processing time and costs.

It can also be seen that the secure requests of one class affects the response times of all classes. For example, changing the security of class 2 non-backup requests from non-secure to secure increased the response times of both classes, [R_1, R_2] from [0.518, 0.974] sec (row 1) to [0.867, 1.346] sec (row 2). This is due to the same reason described above.

Table 4 also shows that the backup options do not have any effect on the response time. This can be seen in the first row where R_1 and R_2 are the same, i.e., [0.518, 0.974] sec, irrespective of the backup options. This is because backup requests, per Assumption A6, do not incur any additional processing time.

Secure requests cannot be backed up because they cannot be sent to the cloud servers (see Assumption A7); thus, the combinations where requests are secure and need backup are shown as "Invalid option" in Table 4.

4.2 Cost Effect

Table 5 clearly shows that the processing costs for secure requests are higher than that of non-secure requests. For example, C_2 is \$0.798/1,000 req for non-secure non-backup requests and \$1.621/1,000 req for secure non-backup requests. Similarly, C_1 is \$0.461/1,000 req for non-secure non-backup requests and \$0.865/1,000 req for secure non-backup requests. This happens because secure requests can only be processed on fog servers, which are not as powerful as cloud servers. Thus, secure requests require more processing time, which in turn increases processing costs.

It can also be seen that changing the security requirements of requests of one class affects the processing costs of all classes of requests. For example, enabling the security of class 2 requests increased the processing costs of both classes, [C_1, C_2] from [0.461, 0.798] to [0.865, 1.621] \$/1000 req. This is due to the same reason described above.

Table 5 also shows that backup requests cost more than non-backup requests. For example, C_2 is \$0.798 (row 1, [nb, nb]) for non-secure non-backup requests and is \$1.358 (row 1, [nb, b]) for non-secure backup requests. Similarly, C_1 is \$0.461 (row 1, [nb, nb]) for non-secure non-backup requests, and is \$0.821 (row 1, [b, nb]) for non-secure backup requests. This is because backup requests incur additional transmission costs for sending the data processed at the fog servers to the cloud servers for backup, and because of the storage cost for backing up data at the cloud servers.

5 *FogQN-AC*: Autonomic Controller

As discussed before, an important consideration in fog computing is the dynamic determination of the optimal fraction f_{opt} of data processing executed at the cloud versus at fog servers. This determination requires that we consider that the processing capacity of fog servers is typically smaller than that of cloud servers. On the other hand, it may be more expensive to use cloud resources as opposed to fog servers. As f increases, more data has to be sent and received from the cloud. On the other hand, fog servers are typically resource-constrained and may not have enough capacity to handle requests from numerous sensors and other IoT devices and may become a bottleneck.

This section presents the design of an autonomic controller, called *FogQN-AC*, that dynamically changes the fraction of data processing performed at the cloud as the workload varies over time. The controller seeks to maintain an optimal or near-optimal value of a utility function that represents stakeholder-defined tradeoffs between average response time and cost. The *FogQN* model described in Section 2 is used to compute the average response time and cost. We implemented *FogQN-AC*, which is publicly available at [14], [15].

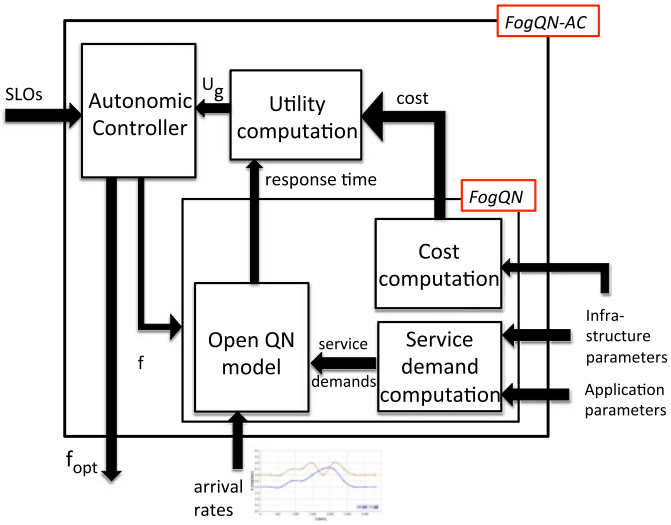

Fig. 3. Architecture of the *FogQN-AC* system.

5.1 Architecture

A high level architecture of the *FogQN-AC* is shown in Fig. 3. The controller monitors the arrival rates λ_r for $r = 1, \ldots, R$ and at regular intervals of time τ determines the optimal value of the vector $\vec{f}$. One component of *FogQN-AC* is the performance model *FogQN* described in Section 2, which provides the response time and cost values to the module that computes the utility function described below. This function is used by the autonomic controller to find an optimal value of $\vec{f}$ that maximizes the utility function subject to Service Level Objectives (SLO).

Although we used a specific optimization algorithm, based on the hill climbing meta-search algorithm, and a particular utility function described in the next section, the architecture/design of the controller is very flexible and can accommodate many other optimal search algorithms or computational optimization techniques, as well as different types of system utility functions.

5.2 The Optimization Problem

The goal of the autonomic controller is to maximize a global utility function U_g defined as a function of the utility function of the average response time and of a utility function of the request processing cost. Utility functions have been commonly used in autonomic computing (see e.g., [16]). We show below the global utility function used here; other definitions are possible. All utility functions defined here have values in [0, 1] such that 1 represents the best possible value and 0 the worst

$$U_g(\vec{R}(\vec{f}), \vec{C}(\vec{f})) = \sum_{r=1}^{R} \omega_r U_{g,r}(R_r(\vec{f}), C_r(\vec{f})), \tag{11}$$

where $\vec{R}(\vec{f}) = (R_1(\vec{f}), \ldots, R_R(\vec{f}))$ is a vector of average response times, $\vec{C}(\vec{f}) = (C_1(\vec{f}), \ldots, C_R(\vec{f}))$ is a vector of costs, $\vec{f} = (f_1, \ldots, f_R)$ is a vector of fraction of data processing at the cloud, $\omega_r > 0 \; \forall \; r$, $\sum_{r=1}^{R} \omega_r = 1$, and $U_{g,r}(R_r(\vec{f}), C_r(\vec{f}))$, given below, is a utility function for class r requests which includes a response time utility function $U_r^{\text{resp}}(R_r(\vec{f}))$ and a cost utility function $U_r^{\text{cost}}(C_r(\vec{f}))$

$$U_{g,r}(R_r(\vec{f}), C_r, (\vec{f})) = w_{\text{resp},r} U_r^{resp}(R_r(\vec{f})) + w_{cost,r} U_r^{cost}(C_r(\vec{f})). \tag{12}$$

The weights $w_{\text{resp},r}$ and $w_{\text{cost},r}$ are positive and add to 1 and are set by the system stakeholders. More formally, the optimization problem dynamically solved by the autonomic controller is

$$\begin{aligned} \max_{\vec{f}} \; & U_g(\vec{R}(\vec{f}), \vec{C}(\vec{f})) \\ \text{s.t. } & f_r \in [f_r^{\min}, f_r^{\max}] && \forall r = 1, \ldots, R \\ & 0 < R_r(\vec{f}) \leq R_r^{\max} && \forall r = 1, \ldots, R \\ & 0 < C_r(\vec{f}) \leq C_r^{\max} && \forall r = 1, \ldots, R \\ & f_r = 0 \;\; if \;\; s_r = s && \forall r = 1, \ldots, R, \end{aligned}$$

where

- $R_r^{\max}$: maximum average response time for class r requests (sec)
- $C_r^{\max}$: maximum processing cost for class r requests ($)
- $f_r^{\min}$: minimum value of f_r. This constraint may be necessary because some request classes may need resources only available in the cloud.
- $f_r^{\max}$: maximum value of f_r. This constraint may be necessary because some request classes may need resources only available at fog servers.

5.3 Design of the Controller

The following notation is used to describe the controller:

- $\vec{u}$: vector with R elements in which each element is a random number uniformly distributed in [0, 1]
- $\vec{\epsilon}_r$: vector with R zero elements except for the rth element, which is equal to ϵ.
- $x \oplus y = \min(x + y, 1)$
- $x \ominus y = \max(x - y, 0)$
- Let $\vec{x} = (x_1, \ldots, x_R)$ and $\vec{y} = (y_1, \ldots, y_R)$: So, we define (1) $\vec{x} \oplus \vec{y} = (x_1 \oplus y_1, \ldots, x_R \oplus y_R)$ and (2) $\vec{x} \ominus \vec{y} = (x_1 \ominus y_1, \ldots, x_R \ominus y_R)$.

The controller wakes up every τ seconds and computes $\vec{f}_{opt}$, i.e., the vector $\vec{f}$ that maximizes the global utility function given the response time, cost, and constraints on $\vec{f}$. Note that τ does not need to be constant but could vary depending on the variability of the arrival rates of each class. Algorithm 1 shows the hill-climbing based search algorithm used by the controller. Other combinatorial search methods (e.g., beam-search and tabu-search) could have been used. In our experience with similar optimization problems, hill climbing provides a good compromise between simplicity and accuracy [17]. The algorithm executes MaxRestarts searches (lines 3–38) that go through at most MaxIterations neighborhoods (lines 15–37). A neighborhood of a given point is computed by Algorithm 2 and is obtained by adding and subtracting a small value ϵ to all R elements of $\vec{f}$. Algorithm 2 invokes function CheckConstraints (Algorithm 3) twice (lines 7 and 12) to check if a candidate point to be added to the neighborhood $\mathcal{N}$ does not violate the cost and response time constraints and keeps the utilization of the fog server and the cloud server below 100 percent.

So, the cardinality of the neighborhood of any point is at most $2R$. Thus, the number of points visited by our heuristic

search is of the order of $\mathcal{O}(R)$ (i.e., $\text{MaxRestarts}\times \text{MaxIterations} \times 2 \times R$). The computational complexity of the solution of the open QN model used by the controller is $\mathcal{O}(R^2)$ [8]. Because the open QN model has to be solved for every point visited by the search, the computational complexity of the controller is $\mathcal{O}(R^3)$.

Algorithm 1. Hill-Climbing Optimization

```
Input: MaxIterations, MaxRestarts
NumIter ← 0;
for t = 1 → MaxRestarts do
  NumIter ← 0;
  /* restart from a random point */
  f⃗ ← u⃗
  for r = 1 → R do
    if s_r = s then
      /* secured requests processed only on fog servers */
      f_r=0
    end if
  end for
  Umax ← 0
  /* Visit at most MaxIterations neighborhoods */
  while NumIter ≤ MaxIterations do
    𝒩 ← Neighborhood (f⃗)
    Searching ← False
    for n⃗ ∈ 𝒩 do
      /* compute the global utility of point n⃗ */
      Util ← U_g(R⃗(n⃗), C⃗(n⃗))
      if Util > Umax then
        /* n⃗ improves the global utility */
        Umax ← Util
        Opt [t].f ← n⃗
        Opt [t].Util ← Umax
        f⃗ ← n⃗
        Searching ← True
      end if
    end for
    if Searching = False then
      /* No improvement found in 𝒩 */
      NumIter = MaxIterations + 1
    else
      /* Go to the next neighborhood */
      NumIter ← NumIter + 1
    end if
  end while
end for
f⃗_opt ← Opt [argmax_{i=1}^{MaxRestarts} {Opt[i].Util}].f
```

We used the following sigmoids as response time and cost utility functions; other appropriate functions could be used

$$U_r^{\text{resp}}(R_r(\vec{f})) = K_{\text{resp},r}\frac{e^{\alpha_r(1-R_r(\vec{f})/R_r^{\max})}}{1+e^{\alpha_r(1-R_r(\vec{f})/R_r^{\max})}}$$
$$U_r^{\text{cost}}(C_r(\vec{f})) = K_{\text{cost},r}\frac{e^{\alpha_c(1-C_r(\vec{f})/C_r^{\max})}}{1+e^{\alpha_c(1-C_r(\vec{f})/C_r^{\max})}},$$

where $K_{\text{resp},r}$ and $K_{\text{cost},r}$ are normalization constants that make the utilities range from 0 to 1, and α_r and α_c are positive numbers that determine the shape (i.e., sharpness) of the utility function. Higher values of the shape factor make the utility go to zero faster. We use $\alpha_r = \alpha_c = 4$ in our experiments. The expressions for $K_{\text{resp},r}$ and $K_{\text{cost},r}$ are obtained by making the value of the utility equal to 1 when the metric (response time or cost) is equal to zero. Therefore

$$K_{\text{resp},r} = K_{\text{cost},r} = \frac{1+e^{\alpha_r}}{e^{\alpha_r}}. \tag{13}$$

The equations above assume that the shape factor α_r is the same for the response time and cost sigmoids. We could as easily assume that these shape factors are not the same.

Algorithm 2. Neighborhood Function

```
Input: f⃗, ϵ
𝒩 ← ∅
for r = 1 → R do
  /* find neighbors for non-secured class requests only */
  if s_r = ns then
    /* check constraints for f⃗ ⊕ ϵ⃗_r */
    if CheckConstraints (f⃗ ⊕ ϵ⃗_r) then
      /* add f⃗ ⊕ ϵ⃗_r to 𝒩 */
      𝒩 ← 𝒩 ⋃ (f⃗ ⊕ ϵ⃗_r)
    end if
    /* check constraints for f⃗ ⊖ ϵ⃗_r */
    if CheckConstraints (f⃗ ⊖ ϵ⃗_r) then
      /* add f⃗ ⊖ ϵ⃗_r to 𝒩 */
      𝒩 ← 𝒩 ⋃ (f⃗ ⊖ ϵ⃗_r)
    end if
  end if
end for
Return 𝒩
```

Algorithm 3. CheckConstraints Function

```
Input: f⃗
/* compute the utilization of the fog server and the cloud */
UtilFog ← Σ_{r=1}^{R} λ_r D_{f,r}
UtilCloud ← Σ_{r=1}^{R} λ_r D_{c,r}
if (UtilFog ≥ 1) or (UtilCloud ≥ 1) then
  Return (False)
end if
/* check response time, cost, and f⃗ constraints */
for r = 1 → R do
  if R_r(f⃗) > R_r^max or C_r(f⃗) > C_r^max or f_r < f_r^min or f_r > f_r^max then
    Return (False)
  end if
end for
Return (True)
```

6 Experiments

The experiments in this paper use the workload characterization methodology for IoT workloads that we presented in [18].

6.1 List of Experiments

We conducted the following six experiments with our heuristic controller (*FogQN-AC*, aka *hc* in the rest of the paper):

TABLE 6
Controller Parameters for All Experiments

τ	500	sec
t	[0, τ, 2τ, 3τ, ...]	sec
$\vec{\lambda}_t$	[varies, varies]	req/sec
$\vec{\omega}$	[0.60, 0.40]	
$\vec{w}_{\text{resp}}$	[0.40, 0.60]	
$\vec{w}_{\text{cost}}$	[0.60, 0.40]	
$\vec{R}^{\max}$	[1.25, 1.25]	sec
$\vec{C}^{\max}$	[0.00060, 0.00100]	$/1000 req
$\vec{f}^{\min}$	[0.20, 0.10]	
$\vec{f}^{\max}$	[0.80, 0.65]	
MaxRestarts	50	
MaxIterations	10	
ϵ	0.01	
α_r, α_c	4	

(1) *hc* versus an optimal controller (*bf*, for brute force hereafter) that examines all points in the search space, and is computationally very expensive, using synthetic arrival rates (see Section 6.2).
(2) *hc* versus the no-controller case (*nc*) using synthetic arrival rates (see Section 6.3).
(3) *hc* versus *nc* with arrival rates coming from a sample of the Google traces [19] (see Section 6.4).
(4) *hc* versus *nc* with arrival rates obtained from a sample of the CityPulse smart city road traffic dataset [4] (see Section 6.5).
(5) *hc* versus an optimization solver using "*A Mathematical Programming Language (AMPL) LGO Solver*" (see Section 6.6).
(6) effect of backup and security options on the utility improvement using *hc* (see Section 6.7).

The following general notes apply to all experiments reported in this paper.

- All the parameters used in the experiments are in Tables 1 (for application, infrastructure, and computed parameters) and 6 (for controller parameters).
- The controller dynamically sets $\vec{f}$ as $\vec{\lambda}$ varies over time
- For the no controller case, $\vec{f}$ is set to the optimal fractions at the starting point and remains constant throughout the experiment.
- The utility improvement ζ using the controller compared to that with no-controller is defined as

$$\zeta = 100 \times \frac{(U_g \text{ with hc} - U_g \text{ without controller})}{U_g \text{ without controller}}. \quad (14)$$

6.2 Autonomic Controller Versus Optimal Search

Table 7 shows in column 1 time variation in increments of 500 sec. The values of the arrival rates λ_1 and λ_2 for each time interval are shown in columns 2 and 3. The table is divided into two parts, each showing, respectively for *hc* and *bf*, the optimal values of f_1 and f_2, average response times R_1 and R_2 for each class, costs C_1 and C_2 per 1,000 requests for each class, and global utility. The last column shows the ratio η between the number of points visited by *hc* and *bf*. As the table indicates, both methods achieve similar utility values with *hc* visiting between 6.7 and 17.5 percent of the points analyzed by *bf*.

Fig. 4 consists of five graphs that compare the variation over time of the following metrics for both classes for *hc* (solid line) and *bf* (dashed line): (1) λ_1 and λ_2, (2) optimal fraction f_1 and f_2, (3) average response times R_1 and R_2, (4) cost per 1,000 requests C_1 and C_2, and (5) global utility U_g. The figure shows that the optimal global utility U_g obtained from both *hc* and *bf* match very closely for all values of t. However, there are minor differences because multiple optimal values of $\vec{f}$ can lead to the same optimal global utility. As a result, there could be small variations in the average response time and cost curves at those time instants. The *hc* controller is very efficient compared to the optimal *bf* one and it visited an average of 11 percent of the points visited by the exhaustive search method. The maximum number of points visited by *hc* is linear with R whereas *bf* is exponential with R (i.e., Δ^R), where Δ is the discretization interval on $\vec{f}$.

TABLE 7
Autonomic Controller (*hc*), Algorithm 1) Versus Optimal Controller (Brute Force, *bf*), Synthetic Workload
(Note: λ in req/sec and C in $/1000 req)

Input Parameters			Autonomic Controller (hc)							Optimal Controller (Brute Force (bf))							
t	λ_1	λ_2	f_1	f_2	R_1	R_2	C_1	C_2	U_g	f_1	f_2	R_1	R_2	C_1	C_2	U_g	η
(sec)			(opt)	(opt)	(sec)	(sec)				(opt)	(opt)	(sec)	(sec)				
0	2.0	3.0	0.229	0.646	0.454	1.225	0.428	0.847	0.7375	0.200	0.650	0.437	1.231	0.425	0.851	0.7379	0.126
500	2.0	3.0	0.214	0.644	0.446	1.225	0.428	0.849	0.7377	0.200	0.650	0.437	1.231	0.425	0.851	0.7379	0.175
1000	2.0	3.0	0.208	0.642	0.444	1.224	0.429	0.849	0.7377	0.200	0.650	0.437	1.231	0.425	0.851	0.7379	0.140
1500	2.5	3.5	0.443	0.645	0.612	1.247	0.487	0.874	0.6899	0.430	0.650	0.603	1.253	0.485	0.876	0.6903	0.107
2000	2.5	3.5	0.503	0.629	0.648	1.227	0.494	0.866	0.6883	0.430	0.650	0.603	1.253	0.485	0.876	0.6903	0.095
2500	3.0	4.0	0.699	0.597	0.792	1.223	0.547	0.894	0.6397	0.610	0.650	0.732	1.274	0.529	0.900	0.6460	0.068
3000	3.5	3.0	0.632	0.581	0.733	1.178	0.518	0.856	0.6780	0.690	0.540	0.771	1.133	0.528	0.845	0.6782	0.087
3500	3.5	4.0	0.775	0.610	0.839	1.233	0.558	0.892	0.6255	0.680	0.650	0.780	1.280	0.545	0.907	0.6296	0.067
4000	2.5	3.5	0.554	0.617	0.679	1.211	0.500	0.859	0.6864	0.430	0.650	0.603	1.253	0.485	0.876	0.6903	0.083
4500	2.0	3.0	0.203	0.618	0.449	1.204	0.438	0.850	0.7368	0.200	0.650	0.437	1.231	0.425	0.851	0.7379	0.173
5000	2.0	3.0	0.432	0.582	0.584	1.150	0.461	0.823	0.7320	0.200	0.650	0.437	1.231	0.425	0.851	0.7379	0.160
5500	2.0	3.0	0.208	0.641	0.444	1.223	0.429	0.849	0.7377	0.200	0.650	0.437	1.231	0.425	0.851	0.7379	0.131

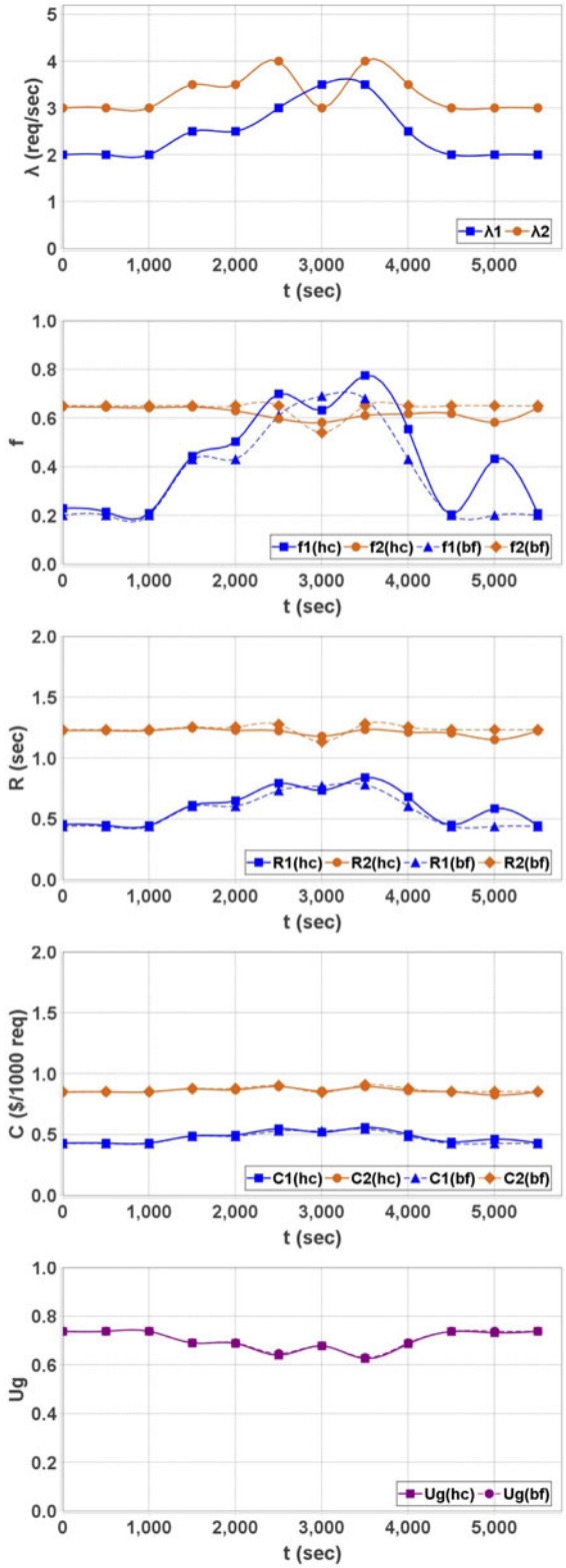

Fig. 4. Autonomic controller (*hc*) versus optimal controller (brute force, *bf*), synthetic workload: plots from top to bottom showing the variation over time of: request arrival rate, fraction of processing at the cloud, average response time, average cost per 1,000 requests, and global utility.

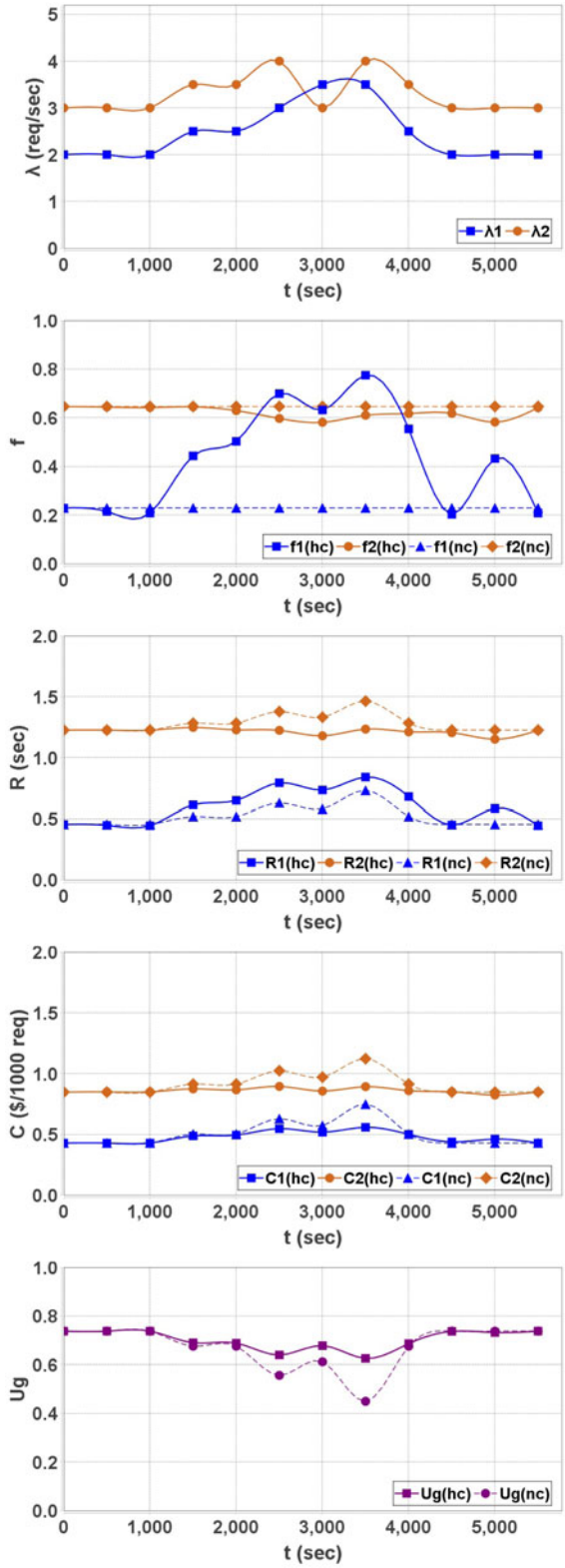

Fig. 5. Autonomic controller (hc) versus no-controller (nc, $\vec{f}$= [0.229, 0.646]), synthetic workload: plots from top to bottom showing the variation over time of: request arrival rate, fraction of processing at the cloud, average response time, average cost per 1,000 requests, and global utility.

6.3 Autonomic Controller Versus No Controller (Synthetic Workload)

This section compares the results obtained with our controller (*hc*) with those using no-controller (*nc*), i.e., a case in which the fractions f_1 and f_2 remain constant at $f_1 = 0.229$ and $f_2 = 0.646$ throughout the experiment. These initial values of f_1 and f_2 are optimal at the beginning of the experiment. The results are shown in Fig. 5, which shows the variation of f_1^{opt} and f_2^{opt} over time. We can see that at time $t = 1500$ sec, the arrival rate of requests starts to increase

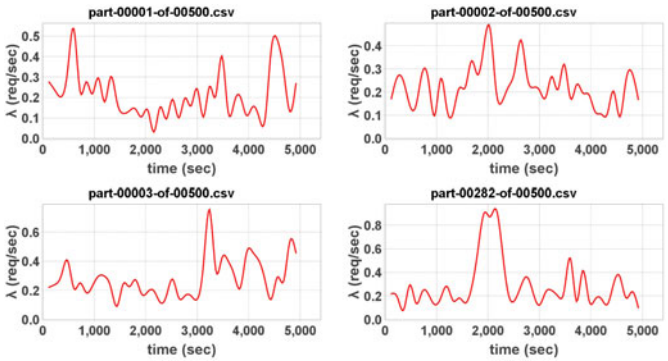

Fig. 6. A few Google cluster-usage traces using a 120-sec bucket interval, λ (req/sec) versus time (sec).

for both classes. As a consequence, the controller starts to send more requests to be processed at the cloud (i.e., f_1^{opt} increases from 0.229 to 0.775). However, f_2^{opt} varies much less (i.e., in the range [0.581, 0.646]). The explanation for the difference in variation of f_1^{opt} and f_2^{opt} is that (1) the arrival rate for class 1 increases from its low value of 2 req/sec to its peak of 3.5 req/sec (i.e., a 75 percent increase) before it starts to decrease. (2) the arrival rate of class 2 requests increases from 3.0 req/sec to 4.0 req/sec (a 33.3 percent increase) in two peaks with an intervening return to its original value. Additionally, class 1 has a higher weight than class 2 in the composition of the utility function (Table 6 indicates that $\omega_1 = 0.6$ and $\omega_2 = 0.4$). The response time and cost curves show that without the controller (dashed lines), the response times and costs exhibit spikes due to the arrival rate increases. Consequently, the global utility U_g, which depends on response time and cost (see Eq. (11)), shows a significant drop without the controller. More specifically, the controller is able to keep the global utility in the range of [0.626, 0.738] while an uncontrolled system exhibits global utilities in the range of [0.449, 0.738]. The utility improvement (ζ) using the controller compared to that with no-controller is highest (39 percent) when the load is highest, i.e., at t = 3500 sec where the arrival rate is in [3.5, 4.0].

6.4 Google Trace Workloads – Autonomic Controller Versus No Controller

This section compares results obtained with our autonomic controller (*hc*) with those using no-controller (*nc*), where the arrival rates for classes 1 and 2 are obtained from the well-known Google cluster-usage trace [19]. This trace consists of workload data on a 12,500 machine cluster over a 29 day period in May 2011. Work arrives to the cluster in the form of jobs described in the *job events table*. The trace consists of 500 CSV files. We extracted all the job submission entries from the files and computed the job interarrival times and, subsequently, workload intensities using a 120-sec bucket intervals. Fig. 6 shows a few examples of Google's workload traces.

The Google experiments used the following traces from Fig. 6 for the arrival rates: (a) class1: first row, second column (file: part_00002_of_00500.csv) and (b) class 2: second row, second column (file: part_00282_of_00500.csv). We applied scaling factors of 2.1 and 5.4 to the arrival rates of classes 1 and 2, respectively, to ensure a reasonably high utilization of the fog and cloud servers at the arrival rate peaks.

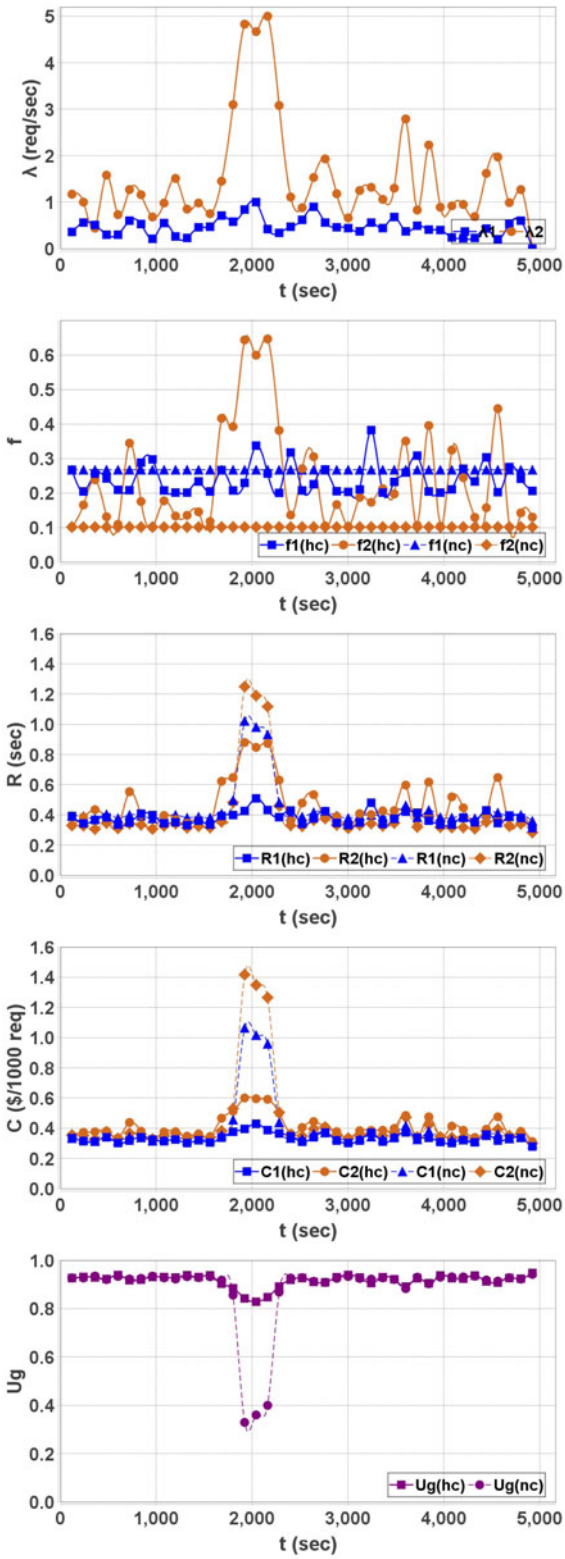

Fig. 7. Google traces – autonomic controller (hc) versus no controller (nc, $\bar{f}$= [0.267, 0.101]) – plots from top to bottom showing the variation over time of: request arrival rate, fraction of processing at the cloud, average response time, average cost per 1,000 requests, and global utility.

All input parameters are the same as before except for λ_r that comes from the Google traces and $L_r = [4.0, 5.0]$ MB; the results are shown in Fig. 7. The curves show that

the controller varies f_1 between 0.2 and 0.382 and f_2 between 0.1 and 0.646. The global utility U_g with the controller enabled is mostly higher than when the controller is inactive. More specifically, for the most part, the controller keeps the utility at around 0.9 (compared to 0.85 without the controller) except at the vicinity of $t = 2040$ sec when the arrival rate of class 2 requests experiences a surge. Then, the utility lowers to 0.829 (compared to 0.36 without the controller).

6.5 CityPulse Smart City Road Traffic Data – Autonomic Controller Versus No Controller

This section compares results obtained with our autonomic controller (*hc*) with those using no-controller (*nc*) when the arrival rates for classes 1 and 2 are obtained from the CityPulse Smart City road traffic data [4]. The data consists of a collection of datasets of vehicle traffic observed at several locations over a period of 6 months (449 observation locations in total). For this experiment, we used vehicle traffic data for Monday, Feb 17th 2014, between 4am and 9pm, from two observation locations (ids = 158324 and 158355) for classes 1 and 2, respectively. For this subset of records, we computed vehicle interarrival times and average arrival rates per hour. We applied a scaling factor of 60 to each of the arrival rates to ensure a reasonably high utilization of the fog servers at the arrival rate peaks. This would be equivalent to a fog server handling traffic for 60 observation points. All input parameters are the same as before except for λ_r that comes from the CityPulse vehicle traffic dataset and $L_r = \{3.0, 4.5\}$ MB; the results are shown in Fig. 8. The curves show that the controller varies f_1 between 0.201 and 0.410 and f_2 between 0.090 and 0.649. At t = 120 sec (i.e., at 6am), the vehicle arrival rates peak for both classes 1 and class 2, and the controller increases f_1 and f_2 to cope with the load. As a result, the global utility U_g with the controller enabled is 0.866, which is significantly higher compared to that without the controller, 0.39. The global utility ranges from 0.866 to 0.962 with the controller enabled, whereas it varies from 0.39 to 0.959 without the controller. The global utility U_g with the controller enabled is mostly higher than that when the controller is inactive (no controller).

6.6 Optimization Using LGO

Although *FogQN-AC* uses a hill climbing approach (*hc*) for finding the fraction ($\vec{f}_{opt}$) that provides optimal or near optimal global utility, it can use various other optimization techniques in lieu of *hc*. The reason could be to balance the tradeoff between a lower computational effort and a potentially acceptable U_g. This section compares the computational effort and results from different optimization techniques.

To analyze such tradeoff, we chose *"A Mathematical Programming Language (AMPL)"*, which is an algebraic modeling language to describe and solve high-complexity problems for large-scale mathematical computing [20]. Out of the many solvers supported by AMPL, we chose *LGO* (Lipschitz Global Optimizer), which can find solutions to global optimization problems that have (possibly many) locally optimal solutions. The types of problems supported by *LGO* are continuous nonlinear objectives and constraints without restriction as to smoothness or convexity [20], [21]. Because *hc* and *LGO* were developed in Java and C, respectively, it may not be appropriate to compare their execution times. Therefore, we used the number of

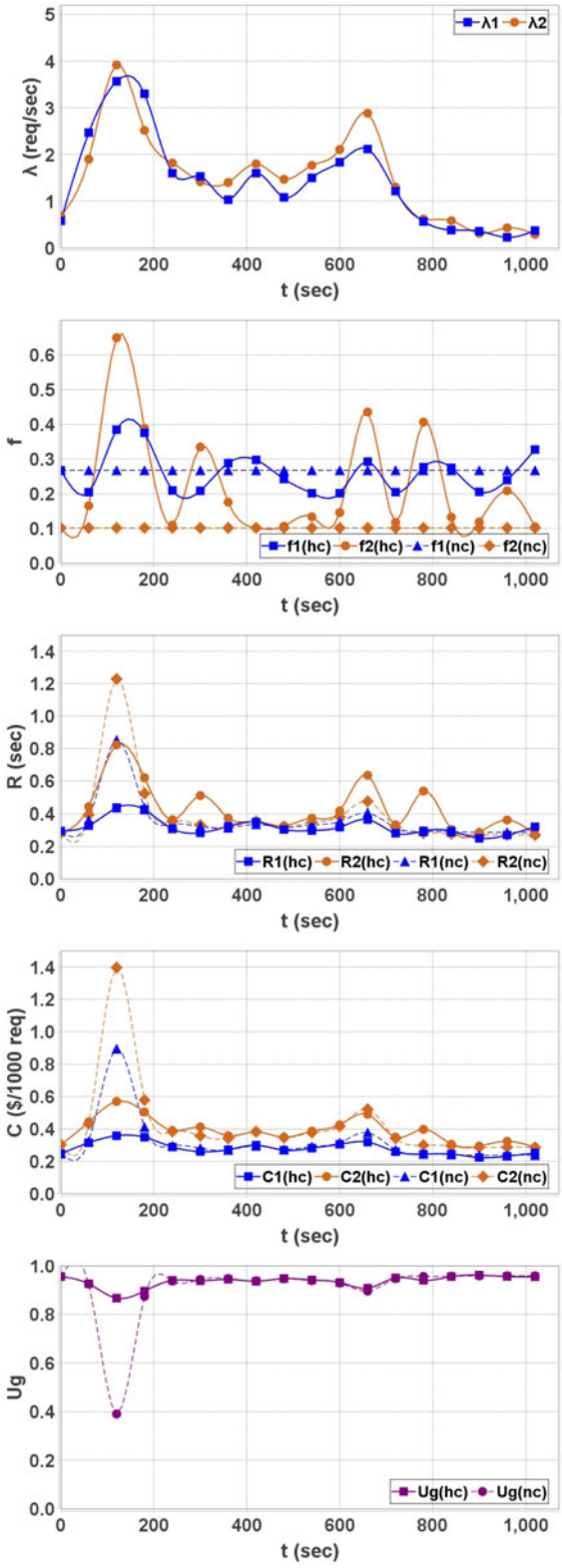

Fig. 8. CityPulse smart city road traffic dataset – autonomic controller (*hc*) versus no controller (*nc*, $\vec{f}$= [0.267, 0.101]) – plots from top to bottom showing the variation over time of: request arrival rate, fraction of processing at the cloud, average response time, average cost per 1,000 requests, and global utility.

TABLE 8
Number of Points Visited by *hc* and by the *LGO* Solver

$\vec{\lambda}$	η	ζ
[0.18, 1.50]	11%	−0.2%
[0.18, 1.50, 1.00]	7%	−0.3%
[0.18, 1.50, 1.00, 0.80]	6%	−1.8%
[0.18, 1.50, 1.00, 0.80, 1.20]	4%	−3.2%
[0.18, 1.50, 1.00, 0.80, 1.20, 2.00]	2%	−3.5%
[0.18, 1.50, 1.00, 0.80, 1.20, 2.00, 1.00]	3%	−3.0%
[0.18, 1.50, 1.00, 0.80, 1.20, 2.00, 1.00, 0.70]	7%	−4.7%
[0.18, 1.50, 1.00, 0.80, 1.20, 2.00, 1.00, 0.70, 0.10]	6%	−3.8%
[0.18, 1.50, 1.00, 0.80, 1.20, 2.00, 1.00, 0.70, 0.10, 0.20]	6%	−5.5%

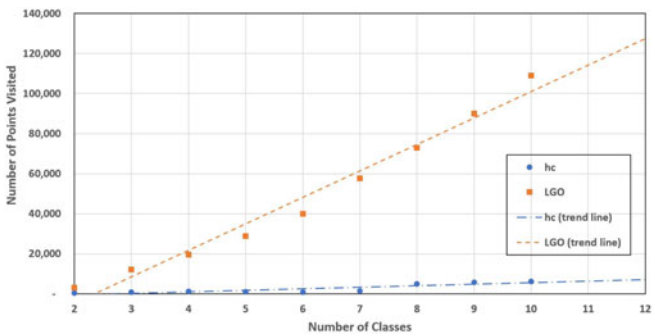

Fig. 9. Comparison of hc with *LGO* solver – Number of Points Visited versus Number of Classes.

points computed/visited as the metric of comparison for computational effort.

We conducted nine sets of experiments with the number of classes varying from 2 to 10; each set consisting of two experiments, one with *hc* and the other with the *LGO* solver with the same input parameters (see Table 8). For *hc*, MaxRestarts varies from 50 for a smaller number of classes (up to 6) to 300 for 8–10 classes. The number of MaxIterations was 10 for all cases. The table shows in columns 2 and 3, respectively, (i) the ratio η between the number of points visited by *hc* and by the *LGO* solver; and (ii) the mild degradation in the U_g with *hc*, which is defined as $\zeta = [U_g \text{ with hc} - U_g \text{ with LGO}]/U_g \text{ with LGO}$.

The global utility U_g for *hc* is slightly lower compared to that with *LGO*, i.e., between −0.2% to −5.5%. However, the number of points visited by *hc* is significantly smaller; in the range of 2 to 11 percent of the points visited by the *LGO* solver. Also, as shown in Fig. 9, the number of points visited by *LGO* grows with the problem complexity (measured by the number of classes) at a much higher rate (about 20 times faster) than with *hc*. Therefore, we can conclude that different techniques and solvers have different computational effort and different U_g's. Thus, the requirements of a given problem may decide the choice of solvers.

6.7 Security and Backup Options on *FogQN-AC*

We conducted six experiments to analyze the effect of security and backup options on *FogQN-AC*. A high level summary of the results of these experiments is shown in Table 9. The maximum value of the utility improvement (ζ) obtained during the experiments varied from 17 percent (for experiment # 1) to 2,356 percent (for experiment # 4).

TABLE 9
Backup and Security Options of Classes of Requests Experiments – Summary of Results

Exp #	Backup: $\vec{b}$	Security: $\vec{s}$	Max(ζ)
1	[b, nb]	[ns, ns]	17%
2	[nb, b]	[ns, ns]	58%
3	[nb, nb]	[s, ns]	123%
4	[nb, nb]	[ns, s]	2,356%
5	[b, nb]	[ns, s]	23%
6	[nb, b]	[s, ns]	48%

The table shows that the controller has accomplished its goal of maximizing the global utility, U_g, with many different combinations of backup and security options of the classes of the requests.

7 Related Work

Fog Computing. Fog computing addresses the problem of high latency and network congestion with cloud computing. This is achieved by extending the cloud computing paradigm to the edge of the network [1], [2], [22]. Fog computing is most appropriate for various critical IoT services and other latency sensitive applications [2]. The architecture of a fog computing environment has hierarchical layers of sensors/actuators, fog servers, and cloud servers [23].

Bittencourt *et al.* presented a general architecture that supports virtual machine migration in fog computing [24]. The work in [25] presented decentralized design patterns on elasticity in IoT and cloud-based systems. The work in [22] presents a survey of fog computing and the authors of [26] present a simulation-based toolkit for fog computing. The work in [27] discusses the challenges of modeling the Internet of Things and the authors of [28] proposed a three-tier architecture for fog computing and used it to evaluate the performance of IoT systems.

The authors of [29] proposed a method for reducing latency and device energy consumption using fog computing and demonstrated that having access to fog resources saves more time and energy than the cloud alone. The work in [30] provides a cloud platform that improves the execution of data-intensive applications by enabling both computation and data movement so that data and tasks can migrate from cloud to edge resources and vice versa. The authors of [31] describe the Secure and Safe Internet of Things project (SerIoT), which optimizes information security in IoT platforms and networks.

The work in [1] analyzes the challenges that occur with fog computing and presents a taxonomy of fog computing according to the identified challenges and its key features. The authors of [32] proposed resource management strategies at each fog node to improve quality of service (QoS). The work in [33] presented an edge-based programming framework that allows users to define a method of processing data streams based on the content and location of the data.

Proper data placement is very important in fog computing because misplaced data can increase network latency, especially when data is shared and used among many nodes. The authors of [34] proposed a fog-aware runtime strategy for placing IoT data in a fog infrastructure that

takes advantage of the heterogeneity and location of fog nodes to minimize overall storage and service latencies. The work in [35] presents a three-layer architecture model for efficient data storage management for edge analytics. This model focuses on reducing the amount of data stored on edge nodes without affecting forecast accuracy.

Predicting user-side quality of service for IoT applications is very hard as it can vary with time and location. The work in [36] proposes IoTPredict, a novel neighborhood-based QoS prediction for candidate services in IoT applications that uses user-contributed QoS information to enable collaborative filtering. Several authors studied the problem of offloading applications to the cloud. The authors of [37] presented an offloading algorithm that employs estimated averages of the execution and communication costs of application modules to decide on a module subset to be offloaded with the objective of minimizing metrics such as execution time. Other examples of offloading studies include [38], [39] and Other very good IoT and fog computing surveys can be found in [40], [41], [42], [43].

The vision and challenges of edge computing were discussed in [2], [44]. In [45], the authors present a case study of edge computing on a prototype based on an open source 3-D arcade game.

The work in [46] aims at reducing the response time of IoT applications by offloading the load of fog-capable devices to the cloud. Fan and Ansari [47] presented an application aware scheme to allocate IoT-based workloads to edge servers in order to minimize the response time of IoT applications. The work in [29] proposes a method for reducing latency and device energy consumption using the fog, which is based on computational offloading and network utility optimization. The work in [48] discusses a human-centered edge-device based computing, known as Edge-centric Computing, and the research challenges associated with its implementation. The work in [49] proposed a new technique called Home Edge Computing, a three-tier edge computing architecture that provides data storage and processing near the users (home server) to achieve ultra-low latency. Pereira *et al.* [50] discuss an experimental evaluation of latency in IoT service composition with mobile gateways and assess the capabilities and limitations of a standard machine-to-machine middleware.

Queuing Network Models. Queuing networks have been extensively used for performance modeling. For example, the authors of [51], [52], [53], [54] used models based on queuing theory to study service performance in cloud computing. The work in [55] developed and analyzed a cloud computing model by applying queuing theory to allocate resources depending upon buffer size in order to increase the performance of a cloud. The authors of [51] use queuing networks to derive performance predictions of a smart parking application. Similarly, the work in [52] uses an M/G/1 queue to model energy consumption and response time tradeoffs for an edge device powered by solar energy that sends messages to some cloud services.

Also, in [54], a combination of M/M/1 and M/M/m queues in sequence was used to model a cloud platform, which was used in tuning service performance, guaranteeing SLA contracts between a client and a service provider. The paper [56] presented an approach for designing cloud computing architectures with QoS, which is based on queuing theory and open Jackson networks. The work in [53] uses an M/M/m queue to model a cloud architecture and showed that the model can be used for multiple servers to reduce the mean queue length and waiting time.

Autonomic Controllers. The design of horizontal or vertical cloud elasticity controllers was considered in [57], [58], [59], [60], [61]. A cost-aware elastic provisioner based on scientific workflow requirements was presented in [57]. The cloud auto-scaling mechanism presented in [62] automatically scales computing instances based on workload information and performance desire. Various auto-scaling strategies are evaluated using log traces from a production Google data center cluster in [58]. The sensitivity of auto-scaling mechanisms to the prediction results is investigated by evaluating the influence of performance prediction accuracy on the auto-scaling actions in [59]. The work in [60], [61], [63] presented an autonomic elasticity controller that predicts the estimated response time under workload surges and computes the minimum number of required resources to meet response time SLAs.

In [64], the authors use a learning automata as a decision-maker to offload incoming dynamic workloads into the edge or cloud servers. They also propose an edge server provisioning approach using a long short-term memory model to estimate the future workload and a reinforcement learning technique to make an appropriate scaling decision. They use simulation to assess their technique.

Summary. To our knowledge, there is no work that applied closed-form queuing analytic models for fog/cloud computing and used these models in the design of an autonomic controller that maximizes a utility function of response time and cost.

8 Concluding Remarks

This paper introduced an analytic model, called *FogQN*, for fog computing environments to analyze the performance and cost of processing requests. This model is based on multi-class open queuing networks and supports various backup and security options of requests. The advantage of an analytic model as opposed to a simulation-based model is that the computation using analytic equations is orders of magnitude faster than running simulations and is generally inexpensive. Therefore, the model can be used by an autonomic controller to dynamically adapt the fraction of requests that are processed at the cloud, as demonstrated in Section 5. In addition, we developed a corresponding tool, also called *FogQN*, which incorporates the model equations, and is publicly available at [5]. The model was validated with the JMT simulation tool using both distribution-based arrival rates and inputs from several publicly available real IoT traces. The errors of all the validation experiments were very small, which indicates the robustness of the equations derived here under more realistic arrival processes. Furthermore, we conducted several experiments to study the effect of various backup and security option combinations, on response times and costs.

This paper presented the design and implementation of an autonomic controller, called *FogQN-AC*, that dynamically changes the fraction of data processing performed at the cloud. The controller seeks to optimize a utility function

of the average response time and cost. This utility function uses the equations of the *FogQN* model, which is an analytic response time and cost model. The controller is assessed against a brute-force optimal solution that performs an exhaustive search to find an optimal set of fractions of processing sent to the cloud. The results show that the controller is very efficient in seeking an optimal or a near optimal solution. Additionally, we conducted experiments to assess the controller against the no controller case (or an inactive controller), i.e., static f, using both synthetic traces and real traces (Google traces and a CityPulse smart city road traffic dataset). The experiments show that the controller is able to maintain a high utility in the presence of wide variations of request arrival rates. Several more experiments were conducted to assess the controller against uncontrolled system using inputs from synthetic traces with various backup and security combinations. All the experiments have shown that the global utility with the controller is equal or higher compared to that in an uncontrolled system and that the controller is able to maintain a high utility even in the presence of workload intensity surges.

REFERENCES

[1] R. Mahmud, R. Kotagiri, and R. Buyya, *Fog Computing: A Taxonomy, Survey and Future Directions*. Singapore: Springer, 2018, pp. 103–130.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.

[3] P. Patel, M. Ali, and A. Sheth, "On using the intelligent edge for IoT analytics," *IEEE Intell. Syst.*, vol. 32, no. 5, pp. 64–69, Sep. 2017, doi: 10.1109/MIS.2017.3711653.

[4] M. I. Ali, F. Gao, and A. Mileo, "CityBench: A configurable benchmark to evaluate RSP engines using smart city datasets," in *Proc. 14th Int. Semantic Web Conf.*, 2015, pp. 374–389.

[5] U. Tadakamalla and D. A. Menascé, "FogQN: A tool for modeling fog computing environments." Accessed: Sep. 15, 2020. [Online]. Available: https://www.cs.gmu.edu/~menasce/fogqn/

[6] U. Tadakamalla and D. A. Menascé, "FogQN: An analytic model for fog/cloud computing," in *Proc. 11th IEEE/ACM Int. Conf. Utility Cloud Comput. Companion*, 2018, pp. 307–313.

[7] U. Tadakamalla and D. A. Menascé, "Autonomic resource management using analytic models for fog/cloud computing," in *Proc. IEEE Int. Conf. Fog Comput.*, 2019, pp. 69–79.

[8] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.

[9] M. Bertoli, G. Casale, and G. Serazzi, "JMT: Performance engineering tools for system modeling," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 4, pp. 10–15, 2009.

[10] I. Grigorik, *High Performance Browser Networking: What Every Web Developer Should Know About Networking and Web Performance*. Newton, MA, USA: O'Reilly, 2015.

[11] Seattle, "Seattle dataset." Accessed: Oct. 23, 2018. [Online]. Available: https://data.seattle.gov/browse?category=Transportation

[12] Chicago data portal. Accessed: Oct. 25, 2018. [Online]. Available: https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew

[13] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "Driving with knowledge from the physical world," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2011, pp. 316–324.

[14] U. Tadakamalla and D. A. Menascé, "FogQN-AC: An autonomic controller for fog computing environments." Accessed: Sep. 15, 2020. [Online]. Available: https://www.cs.gmu.edu/~menasce/fogqn-ac/

[15] U. Tadakamalla and D. A. Menascé, "FogQN-AC: User's guide of an autonomic controller for fog computing environments." Accessed: Sep. 15, 2020. [Online]. Available: https://www.cs.gmu.edu/~menasce/fogqn-ac-guide/

[16] M. N. Bennani and D. A. Menascé, "Resource allocation for autonomic data centers using analytic performance models," in *Proc. Int. Conf. Autom. Comput.*, 2005, pp. 229–240.

[17] J. M. Ewing and D. A. Menascé, "A meta-controller method for improving run-time self-architecting in SOA systems," in *Proc. 5th ACM/SPEC Int. Conf. Perform. Eng.*, 2014, pp. 173–184.

[18] U. Tadakamalla and D. A. Menascé, "Characterization of IoT workloads," in *Proc. Int. Conf. Edge Comput.*, 2019, pp. 1–15.

[19] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format + schema," Google Inc., Mountain View, CA, USA, revised 2014–11-17 for version 2.1, Nov. 2011. [Online]. Available: https://github.com/google/cluster-data

[20] AMPL, "AMPL: A mathematical programming language." Accessed: Sep. 15, 2020. [Online]. Available: https://www.ampl.com

[21] AMPL, "AMPL, LGO Solver." Accessed: Sep. 15, 2020. [Online]. Available: https://ampl.com/products/solvers/solvers-we-sell/lgo

[22] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proc. Workshop Mobile Big Data*, 2015, pp. 37–42.

[23] D. Rahbari and M. Nickray, "Scheduling of fog networks with optimized knapsack by symbiotic organisms search," in *Proc. 21st Conf. Open Innov. Assoc.*, 2017, pp. 278–283.

[24] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana, "Towards virtual machine migration in fog computing," in *Proc. 10th Int. Conf. P2P Parallel Grid Cloud Internet Comput.*, 2015, pp. 1–8.

[25] V. Cardellini, T. G. Grbac, M. Nardelli, N. Tanković, and H.-L. Truong, "QoS-Based elasticity for service chains in distributed edge cloud environments," in *Autonomous Control for a Reliable Internet of Services*. Berlin, Germany: Springer, 2018, pp. 182–211.

[26] H. Gupta, A. Dastjerdi, S. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things," *J. Softw. Practice Experience*, vol. 6, pp. 1275–1296, 2017.

[27] G. Kecskemeti, G. Casale, D. N. Jha, J. Lyon, and R. Ranjan, "Modelling and simulation challenges in Internet of Things," *IEEE Cloud Comput.*, vol. 4, no. 1, pp. 62–69, Jan./Feb. 2017.

[28] W. Li *et al.*, "System modelling and performance evaluation of a three-tier cloud of things," *Future Gener. Comput. Syst.*, vol. 70, pp. 104–125, 2017.

[29] S. Ahn, M. Gorlatova, and M. Chiang, "Leveraging fog and cloud computing for efficient computational offloading," in *Proc. IEEE MIT Undergraduate Res. Technol. Conf.*, 2017, pp. 1–4.

[30] D. Bermbach *et al.*, "DITAS: Unleashing the potential of fog computing to improve data-intensive applications," in *Proc. Eur. Conf. Service-Oriented Cloud Comput.*, 2018, pp. 154–158.

[31] J. Domanska, E. Gelenbe, T. Czachorski, A. Drosou, and D. Tzovaras, "Research and innovation action for the security of the Internet of Things: The SerIoT project," in *Proc. Int. ISCIS Secur. Workshop*, 2018, pp. 101–118.

[32] J. Li, C. Natalino, D. P. Van, L. Wosinska, and J. Chen, "Resource management in fog-enhanced radio access network to support real-time vehicular services," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput.*, 2017, pp. 68–74.

[33] E. G. Renart, J. Diaz-Montes, and M. Parashar, "Data-driven stream processing at the edge," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput.*, 2017, pp. 31–40.

[34] M. I. Naas, P. R. Parvedy, J. Boukhobza, and L. Lemarchand, "iFogStor: An IoT data placement strategy for fog infrastructure," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput.*, 2017, pp. 97–104.

[35] I. Lujic, V. D. Maio, and I. Brandic, "Efficient edge storage management based on near real-time forecasts," in *Proc. IEEE 1st Int. Conf. Fog Edge Comput.*, 2017, pp. 21–30.

[36] G. White, A. Palade, C. Cabrera, and S. Clarke, "IoTPredict: Collaborative QoS prediction in IoT," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2018, pp. 1–10.

[37] X. Gao, X. Huang, S. Bian, Z. Shao, and Y. Yang, "PORA: Predictive offloading and resource allocation in dynamic fog computing systems," *IEEE Internet of Things J.*, vol. 7, no. 1, pp. 72–87, Jan. 2020.

[38] H. Haridas, S. Kailasam, and J. Dharanipragada, "Cloudy knapsack algorithm for offloading tasks from large scale distributed applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 4, pp. 949–963, Fourth Quarter 2019.

[39] J. Barrameda and N. Samaan, "A robust formulation for efficient application offloading to clouds," *IEEE Trans. Cloud Comput.*, vol. 8, no. 3, pp. 710–720, Third Quarter 2020.

[40] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.
[41] M. A. A. da Cruz, J. J. P. C. Rodrigues, J. Al-Muhtadi, V. V. Korotaev, and V. H. C. de Albuquerque, "A reference model for Internet of Things middleware," *IEEE Internet of Things J.*, vol. 5, no. 2, pp. 871–883, Apr. 2018.
[42] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.
[43] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, "IoT middleware: A survey on issues and enabling technologies," *IEEE Internet of Things J.*, vol. 4, no. 1, pp. 1–20, Feb. 2017.
[44] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[45] G. Premsankar, M. D. Francesco, and T. Taleb, "Edge computing for the Internet of Things: A case study," *IEEE Internet of Things J.*, vol. 5, no. 2, pp. 1275–1284, Apr. 2018.
[46] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue, "On reducing IoT service delay via fog offloading," *IEEE Internet of Things J.*, vol. 5, no. 2, pp. 998–1010, Apr. 2018.
[47] Q. Fan and N. Ansari, "Application aware workload allocation for edge computing-based IoT," *IEEE Internet of Things J.*, vol. 5, no. 3, pp. 2146–2153, Jun. 2018.
[48] P. Garcia Lopez *et al.*, "Edge-centric computing: Vision and challenges," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 5, pp. 37–42, Sep. 2015.
[49] C. S. M. Babou, D. Fall, S. Kashihara, I. Niang, and Y. Kadobayashi, "Home edge computing (HEC): Design of a new edge computing technology for achieving ultra-low latency," in *Proc. Int. Conf. Edge Comput.*, 2018, pp. 3–17.
[50] C. Pereira, A. Pinto, D. Ferreira, and A. Aguiar, "Experimental characterization of mobile IoT application latency," *IEEE Internet of Things J.*, vol. 4, no. 4, pp. 1082–1094, Aug. 2017.
[51] T. Bures, V. Matena, R. Mirandola, L. Pagliari, and C. Trubiani, "Performance modelling of smart cyber-physical systems," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.*, 2018, pp. 37–40.
[52] P. G. Harrison and N. M. Patel, "Optimizing energy-performance trade-offs in solar-powered edge devices," in *Proc. ACM/SPEC Int. Conf. Perform. Eng.*, 2018, pp. 253–260.
[53] T. S. Sowjanya, D. Praveen, K. Satish, and A. Rahiman, "The queueing theory in cloud computing to reduce the waiting time," *Int. J. Comput. Sci. Eng. Technol.*, vol. 1, no. 3, pp. 110–112, 2011.
[54] J. Vilaplana, F. Solsona, I. Teixidó, J. Mateo, F. Abella, and J. Rius, "A queuing theory model for cloud computing," *J. Supercomput.*, vol. 69, no. 1, pp. 492–507, 2014.
[55] P. S. Varma, A. Satyanarayana, and M. V. R. Sundari, "Performance analysis of cloud computing using queuing models," in *Proc. Int. Conf. Cloud Comput. Technol. Appl. Manage.*, 2012, pp. 12–15.
[56] J. Vilaplana, F. Solsona, and I. Teixidó, "A performance model for scalable cloud computing," in *Proc. 13th Autralasian Symp. Parallel Distrib. Comput.*, 2015, pp. 51–60.
[57] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster, "Cost-aware cloud provisioning," in *Proc. IEEE 11th Int. Conf. e-Sci.*, 2015, pp. 136–144.
[58] M. A. Netto, C. Cardonha, R. L. Cunha, and M. D. Assunçao, "Evaluating auto-scaling strategies for cloud computing environments," in *Proc. 22nd Int. Symp. Modelling Anal. Simul. Comput. Telecommun. Syst.*, 2014, pp. 187–196.
[59] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Measuring prediction sensitivity of a cloud auto-scaling system," in *Proc. IEEE 38th Int. Comput. Softw. Appl. Conf. Workshops*, 2014, pp. 690–695.
[60] V. Tadakamalla and D. A. Menascé, "Analysis and autonomic elasticity control for multi-server queues under traffic surges," in *Proc. Int. Conf. Cloud Autonomic Comput.*, 2017, pp. 92–103.
[61] V. Tadakamalla and D. A. Menascé, "An analytic model of traffic surges for multi-server queues in cloud environments," in *Proc. IEEE 11th Int. Conf. Cloud Comput.*, 2018, pp. 668–677.
[62] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," in *Proc. 11th IEEE/ACM Int. Conf. Grid Comput.*, 2010, pp. 41–48.
[63] V. Tadakamalla and D. A. Menascé, "Model-driven elasticity control for multi-server queues under traffic surges in cloud environments," in *Proc. IEEE Int. Conf. Autonomic Comput.*, 2018, pp. 157–162.
[64] A. Shahidinejad and M. Ghobaei-Arani, "Joint computation offloading and resource provisioning for edge-cloud computing environment: A machine learning-based approach," *Softw., Practice Experience*, vol. 50, no. 12, pp. 2212–2230, 2020.

Uma Tadakamalla (Member, IEEE) received the MS and PhD degrees in computer science from George Mason University, Fairfax, Virginia, in 1995 and 2019, respectively. She is currently a principal software systems architect with CACI, supporting U.S. federal government projects. She has 25 years of experience in software engineering, software systems architecture, and information technology. She also worked with AT&T, Oracle, Nextel/Sprint, Global One, and QinetiQ on various commercial and U.S. federal government projects. She is a recipient of CACI's company-wide 2020 Innovator Award, and Sprint's 2006 Annual Circle of Excellence Award (≈top 0.1 percentile). Her research was published in several distinguished IEEE, ACM, and Springer international conferences. Her research interests include autonomic computing, cloud computing, and data mining.

Daniel A. Menascé (Fellow, IEEE) received the PhD degree in computer science from the UCLA, Los Angeles, California, in 1978. He is a university professor of computer science with George Mason University where he was senior associate dean of the Volgenau School of Engineering for seven years. He is a fellow of the ACM, a recipient of the 2017 Outstanding Faculty Award from the Commonwealth of Virginia, and the recipient of the 2001 lifetime A.A. Michelson Award from the Computer Measurement Group. He is the author of more than 275 papers and five books on analytic models of computer systems published by Prentice Hall. His research interests include autonomic computing, security performance tradeoffs, analytic modeling of computer systems, and software performance engineering.