This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

FogSpot: Spot Pricing for Application Provisioning in Edge/Fog Computing

Argyrios G. Tasiopoulos*, Onur Ascigil*, Ioannis Psaras*, Stavros Toumpis†, George Pavlou*
*Dept. of Electronic and Electrical Engineering, University College London
†Dept. of Informatics, Athens University of Economics and Business
Email: {argyrios.tasiopoulos, o.ascigil, i.psaras, g.pavlou}@ucl.ac.uk, toumpis@aueb.gr

***Abstract*—An increasing number of Low Latency Applications (LLAs) in the entertainment, IoT, and automotive domains require response times that challenge the traditional application provisioning using distant Data Centres. The fog computing paradigm extends cloud computing at the edge and middle-tier locations of the network, providing response times an order of magnitude smaller than those that can be achieved by the current "client-to-cloud" network model. Here, we address the challenges of provisioning heavily stateful LLA in the setting where fog infrastructure consists of third-party computing resources, *i.e.,* cloudlets, that comes in the form of "*data centres in the box*".**

We introduce FogSpot, a charging mechanism for on-path, on-demand, application provisioning. In FogSpot, cloudlets offer their resources in the form of Virtual Machines (VMs) via markets, collocated with the cloudlets, that interact with forwarded users' application requests for VMs in real time. FogSpot associates each cloudlet with a price based on current applications' demand. The proposed mechanism's design takes into account the characteristics of cloudlets' resources, such as their limited elasticity, and LLAs' attributes, like their expected QoS gain and engagement duration. Lastly, FogSpot guarantees the end users' requests truthfulness while focusing in maximising either each cloudlet's revenue or resource utilisation.

I. Introduction

Cloud computing has been a tremendous success in enabling computationally intensive applications to elastically cope with changes in demand for computing resources in a scalable manner, through on-demand computation. Cloud providers mostly focus on the challenging problem of allocating computing resources to different users while exploiting economies of scale for decreasing their running cost [17]. Variants of usage-based pricing schemes are deployed for controlling the demand of cloud resources [8], where users are charged a static per-unit price, *e.g.,* x dollars per hour/workload. However, an increasing number of applications (*e.g.,* augmented reality, automotive, and health monitoring systems) require low response times, rendering the current cloud-based infrastructure unfit for this purpose. We refer to these as Low Latency Applications (LLAs), since they strongly rely on the latency of the network for achieving a satisfying Quality of Service (QoS).

As a result, there is a pressing need for alternative computing infrastructures that augment and complement the typically centralised cloud computing paradigm in order to enable LLAs. Fog computing extends cloud computing at the edge and middle-tier locations of the network by utilising existing in-network computing resources. In this work, we focus on the setting where fog infrastructure consists of third-party

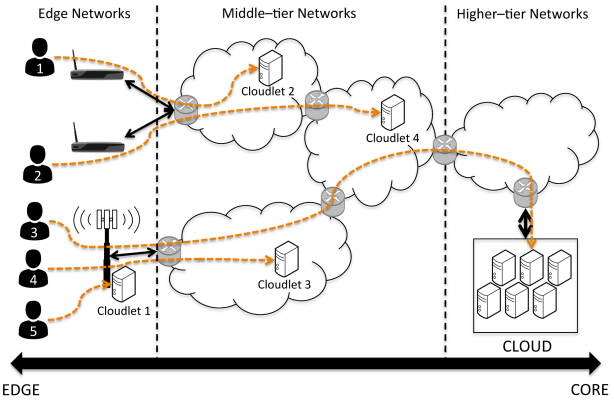


Fig. 1: End-users on-path application request service by cloudlets at the edge and middle-tier locations of the network.

cloudlets that can be deployed as "*data centres in a box*" [26]. In detail, cloudlets can be placed throughout the network, in order to bring computing resources *closer* to the end users, and so have the potential of improving the performance of LLAs by decreasing the corresponding Round-Trip Time (RTT) between end-users and the computing resources they employ.

Nevertheless, cloudlets are incapable of providing the essentially boundless elasticity in computing resources that is currently offered by the cloud. That is, at any given cloudlet the demand of resources can well exceed their availability. Clearly, using a resource allocation mechanism for LLAs that explicitly takes into account the inherently bounded cloudlet elasticity, is of paramount importance. Similarly to the cloud, a pricing scheme can affect the demand levels for cloudlet resources; however, dealing with short-term demand fluctuations can be a challenge [18]. We argue that in order to capture demand fluctuations, cloudlets have to deploy flexible pricing schemes where the price of resources is formed by the current demand of the market.

In this work, *we investigate the emerging market of provisioning heavily stateful LLAs over the fog infrastructure*. In our context, heavily stateful LLAs are applications that cannot be migrated to another cloudlet location, due to the size of their user-generated runtime data, without seriously damaging their QoS. In other words, heavily stateful LLAs cannot be suspended for serving another request. In the setting we consider, computation is available firstly at the edge, secondly at middle-tier locations of the network, in the form of cloudlets, and, finally, at distant clouds/data centres

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

(Fig. 1). We argue that service provisioning over cloudlets is expected to take place in a decentralised and uncoordinated environment. Given that cloudlet resources are limited, the key challenge is to create a market that operates on a per-request basis for offering the finest possible resource allocation granularity. We aim to provide answers to the fundamental questions of: *i*) *how should the cloudlet resources be allocated over time to different applications/services?*, and *ii*) *how much should a cloudlet charge an application?*[1] Here, we present a decentralised pricing mechanism that answers both questions, while addressing the challenges of dynamic service provisioning over the fog computing infrastructure.

Our starting point is the spot pricing mechanism of [2], which creates an auction-based market for available cloud computing resources. In that work, cloud providers determine their spot price at regular time intervals subject to their resources' demand. Then, at each time-interval, user requests that bid above the spot price are accepted while the rest of the users are suspended until the spot price falls below their bid. Clearly, the spot pricing distributed solution to the problem of allocating cloud resources is suitable for tasks that can be disrupted; hence, typical spot pricing is unfit for LLA provisioning since interruption affects their QoS.

For this reason, we introduce FogSpot, a pricing mechanism that associates each cloudlet with a price for *on-path*, on-demand, distributed LLA provisioning. In FogSpot, cloudlets offer their resources in the form of Virtual Machines (VMs) via collocated markets. By on-path provisioning we mean that as LLA requests are forwarded towards a default execution location (*i.e.,* the cloud), they interact with cloudlet markets along the path followed by them. If the price of a market is less than the estimated QoS gain a LLA has, when served by the corresponding cloudlet, an available VM is allocated to serve the request; otherwise, the request is rejected and continues its journey towards the cloud. For example in Fig. 1 the request of user 2 first interacts with cloudlet 2 and is rejected; the request then continues its journey to cloudlet 4, which finally accepts it. On the other hand, the request of user 3 fails to get served by cloudlets 1, 3 and 4, reaching its final execution location at the cloud. FogSpot addresses explicitly the heavily stateful LLA requirement of undisrupted users' engagement to LLA instances while setting the FogSpot price of each cloudlet with the aim of maximising either its revenue or utilisation. To this end, the main technical contributions of this paper are as follows:

1) We study the emerging market of LLA provisioning in fog computing from an economics point of view.
2) We develop FogSpot, a pricing mechanism for on-path, on-request LLA provisioning with respect to cloudlets' computing resource limitations and end users' requirements for undisrupted LLA engagement.
3) We illustrate the merits of FogSpot in realistic topologies in comparison with state-of-the-art proactive and reactive provisioning approaches.

[1]In fact, the application producer/creator.

II. Related Work

A. Cloud Provisioning Pricing

A few existing works consider the problem of cloud resource allocation from a purely scheduling perspective (*e.g.,* see [13] and references therein) while others also include the dimension of pricing for maximising either the cloud provider's revenue [31] or the social welfare of the system [33]. That said, the limited resource elasticity of cloudlets renders existing cloud pricing plans insufficient for direct application.

In today's, cloud computing paradigm, resources are offered in terms of remote instances, *i.e.,* virtual machines (VMs), with dedicated CPU, memory, and storage resources, in the form of Infrastructure-as-a-Service (IaaS) [6]. In particular, Amazon offers instances under three pricing schemes:

1) *Reserved instances* are guaranteed to be available on a long term, *i.e.,* for more than a year, by charging fixed usage-based prices.
2) *On-demand instances* are guaranteed to be available on a short term, *i.e.,* for an hour, by also charging fixed usage-based prices.
3) *Spot instances* are created using an auction-based market for *spare* instances. More specifically, users can use spot instances only if their bids exceed a spot price while spot prices are updated every 5 minutes; this means that the availability of instances is not guaranteed.

Offering cloudlet resources as reserved or on-demand instances fails to capture demand fluctuations under cloudlet's bounded resources elasticity, *i.e.,* requests are rejected even if under-utilised instances exists as opposed to data centres where additional resources are allocated. On the other hand, spot instances are suitable for interruptible jobs, since a task is executed as long as its bid exceeds the current spot price; otherwise, the task is suspended. Spot instance interruptions can affect severely the QoS of LLAs. Despite some efforts in strategically bidding spot cloud instances by taking into consideration the probability of disrupting a task [35], we suggest that *a cloudlet pricing plan should guarantee the undisrupted execution of applications in its inherent characteristics*.

B. Fog Computing & Cloudlets

Fog computing extends the cloud computing paradigm to the edge of the network, providing low latencies that enable the deployment of critical LLAs in the IoT (*e.g.,* health monitoring), entertainment (*e.g.,* Virtual/Augmented reality), and automotive industries (*e.g.,* self-driving cars) [7]. In this work, we focus on the realistic setting where the fog infrastructure consists of third-party computing resources, *i.e.,* cloudlets, interested in offering their resources for provisioning applications. Cloudlets have been proposed in [26] with the initial purpose of acting as a surrogate infrastructure where mobile devices can offload intensive tasks to complement their computing power and battery limitations [4], [12]. Hence, numerous works focus on task offloading technologies [10] with the aim of augmenting the mobile devices' computing capabilities [25], [30] and/or battery duration [5].

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

Closer to our setting, [3] introduces *on-path provisioning* for addressing the problem of application-specific task offloading over the fog infrastructure, *i.e.,* task requests from hosts are simply routed towards the back-end cloud and are opportunistically executed by the cloudlets along the path as demonstrated in Fig. 1. In particular, applications are provisioned according to established techniques in content caching, such as Least-Frequently-Used (LFU), in an uncoordinated fashion. However in [3], the economic aspects of the problem are not considered while the presented caching approaches are suitable for stateless applications whose requests can be decomposed to individual tasks that can be executed by any available application instance in the fog. Initial efforts in on-path pricing has been presented in [29] without however addressing sufficiently the problem of price derivation. On the other hand, [28] presents a market design for the pricing of application provisioning at the Edge under the presence of user mobility, which however is not suitable for the heavily stateful application setting we consider here. Lastly, in [20] the authors propose a self-tuning service/application provisioning combinatorial auction mechanism where application providers bid periodically for a specific number of VMs organised in different execution zones. Nevertheless, the presented mechanism relies on precise predictions about the future demand of an application while the optimal bidding derivation is a computationally expensive process that challenges the execution frequency of the mechanism.

To the best of our knowledge, FogSpot is the first work that addresses in detail the problem of on-demand, distributed, heavily stateful LLA provisioning from an economic perspective over independent fog computing resources.

III. Design Rationale & System Model

In this section, at first, we justify the design choices of the provisioning mechanism we envision. Then, we introduce the system model.

A. Design Rationale

The challenge here is the design of a market mechanism tailored to the provisioning of LLAs over an uncoordinated cloudlet infrastructure. Our design has to address explicitly the challenges of *i*) how the applications should discover the the cloudlets' resources, and *ii*) the cloudlets' limited elasticity.

In our context, cloudlet resource discovery is referred to the process of finding appropriate computing resources for provisioning LLAs. Resource discovery over a set of geographically distributed clouds is a challenging task [14], that is closely associated to the process of resource monitoring [1], for taking full advantage of the cloud's capability of allocating and releasing resources on-demand [34]. In our setting, cloudlet points of presences are expected to exceed by far the number of clouds. Therefore, discovering and allocating resources, as a response to a continuous monitoring process, would face scalability issues.

Given that LLA requests are forwarded in the network towards a distant Data Centre, we argue for both *on-path and on-demand application provisioning*. In particular, by applying *on-path provisioning*, there is no need for a centralised resource discovery process, since resources are discovered in real time. Furthermore, applications are not required to monitor the usage of cloudlets' resources since the *on-demand provisioning* guarantees their full utilisation, *i.e.,* each instance remains active only throughout a user's engagement. These design choices enable the most promising and incrementally deployable conditions for the problem of LLA provisioning we tackle, providing an uncoordinated and distributed solution similar to the content delivery via *on-path* caching [22].

In this work we argue that *cloudlet pricing schemes should follow a pay-as-you-go structure in terms of application user engagement duration* in order to complement and promote the on-demand provisioning of applications. In other words, the LLA providers should be charged based on the time their users occupy a cloudlet instance (*e.g.,* π dollars per 1 ms).

TABLE I: FogSpot Notation

System Model	
$\mathcal{S}$	Set of LLAs.
$\mathcal{D}$,$\tilde{D}_s$	Set of cloudlets, default data centre of LLA s.
μ_s	Exponential distribution engagement duration rate of application s.
$\mathcal{P}$	Set of requests' classes.
$\mathcal{D}_p$	Set of cloudlets serving class p.
$\mathcal{P}_d$	Set of requests' classes arriving at cloudlet d.
$\lambda_{s,p}$	Poisson process arrival rate of LLA s class's p requests.
$u^d_{s,p}$	Per second QoS gain of LLA s at cloudlet d for class p, in terms of network condition.
$J_s(\cdot)$	Average aggregated utility of LLA s .
$y^d_{s,p}$,y_s	Request provisioning rate of LLA s class p at d, provisioning rate matrix of LLA s.
C_d	Number of VMs at cloudlet d.
ρ_{thres}	Target utilisation level of resources.
FogSpot On-Demand Provisioning	
π_d	Price of cloudlet d.
FogSpot Spot Price Derivation Mechanism	
$\mathcal{P}_{d,s}(\pi^t_d)$	Set of classes of LLA s with gain in being provisioned at price π^t_d at cloudlet d.
$X^d_s(t)$	Arrival rate of LLA s at d during iteration t that can be provisioned.
$Y^d_s(t)$	Admitted request rate of LLA s by cloudlet d at iteration t.

B. System Model

We consider a set $\mathcal{S} \triangleq \{1, 2, ..., S\}$ of LLAs and a set $\mathcal{D} \triangleq \{1, 2, ..., D\}$ of cloudlets. The engagement duration of an LLA $s \in \mathcal{S}$ has an exponential distribution with rate parameter μ_s, where $1/\mu_s$ is its mean engagement time. We assume that each LLA can be provisioned at a distant cloud/data centre whose capacity is sufficient, in all cases, for serving the total number of LLA requests it receives. In particular, each LLA request is forwarded via a *path* from the access point of an end user to a default application cloud/data centre, $\tilde{D}_s$ $\forall s \in \mathcal{S}$. Each path p is considered as a distinct traffic class traversing a set of $\mathcal{D}_p$ cloudlets. Specifically, a class p request of LLA s experiences a QoS improvement $u^d_{s,p}$, in terms of network conditions, for each second that the end user remains engaged to an LLA instance at cloudlet $d \in \mathcal{D}_p$, as opposed to the default Data Centre $\tilde{D}_s$. We denote that set of request classes in the network by $\mathcal{P} \triangleq \{1, 2, ..., P\}$. Note that a cloudlet d receives traffic from a set of classes, $\mathcal{P}_d$; that is, the gain of a request of LLA s served at d is class-specific, *i.e.,* two requests of the same LLA but of different classes experience different gains. In our system, we assume that the LLA requests of each class are generated according to a Poisson process of rate $\lambda_{s,p}$ requests per second $\forall s \in \mathcal{S}$ and $\forall p \in \mathcal{P}$.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

In our setting each request concerns the allocation of a single VM. Let the requests of LLA s of class p be admitted at cloudlet d according to a rate $y^d_{s,p}$; we define the *provisioning rate matrix* of LLA s, $y_s \triangleq (y^d_{s,p} : p \in \mathcal{P}, d \in \mathcal{D})$. Note that if $d \notin \mathcal{D}_p$ then $y^d_{s,p} = 0$ for all $s \in \mathcal{S}$. The admitted request rates of a class respects its Poisson generation process rate, *i.e.,* cloudlets $\mathcal{D}_p$ cannot admit more than the generated requests of class p:

$$\sum_{d \in \mathcal{D}_p} y^d_{s,p} \leq \lambda_{s,p}. \tag{1}$$

Note that constraint (1) implies that the default execution location of LLA s, $\tilde{D}_s$, admits requests according to a Poisson process of rate $\lambda_{s,p} - \sum_{d \in \mathcal{D}_p} y^d_{s,p}$, when the admission rates $y^d_{s,p}$ characterise a Poisson process too. We use $J_s(y_s)$, $\forall s \in \mathcal{S}$, to denote the utility of LLA s as a function of y_s in terms of aggregated per-second QoS gain, estimated according to:

$$J_s(y_s) = \frac{1}{\mu_s} \sum_{p \in \mathcal{P}} \sum_{d \in \mathcal{D}_p} u^d_{s,p} y^d_{s,p}, \tag{2}$$

where $y^d_{s,p}/\mu_s$ is the average number of active class p request s sessions at cloudlet $d \in \mathcal{D}_p$, *i.e.,* the average number of allocated VMs to class p of LLA s, by Little's Law.

Let C_d be the number of identical VMs, in terms of dedicated CPU, memory, and storage resources, that cloudlet $d \in \mathcal{D}$ can support.[2] Observe that the average utilised VMs have to respect the VMs' number restriction:

$$\sum_{s \in \mathcal{S}} \frac{1}{\mu_s} \sum_{p \in \mathcal{P}_d} y^d_{s,p} \leq \rho_{\text{thres}} C_d, \ \forall d \in \mathcal{D}, \tag{3}$$

where ρ_{thres} is the target utilisation level of the resources taking values in $[0, 1]$.

IV. FogSpot Mechanism

In this section, we describe the individual components of FogSpot mechanism, namely *i*) on-demand provisioning, and *ii*) price derivation, before concluding with a detailed scenario that demonstrates their operation. Given a price assigned to each market, on-demand provisioning component governs the real-time interaction of forwarding LLA requests and each cloudlet's collocated market. On the other hand, the price derivation component is responsible for assigning a price for each market, which determines the portion and class of requests served at each cloudlet, with the aim of maximising either the cloudlet's revenue or its resource utilisation.

A. FogSpot: On-Demand Provisioning Component

An overview of the FogSpot on-demand provisioning scheme is depicted in Fig. 2, where we consider an end user request of LLA $s \in \mathcal{S}$ that arrives at the market of cloudlet $d \in \mathcal{D}$ (step a). The interaction of a request with each on-path market is characterised by the following properties:

[2]The presented mechanism could be applied over different types of instances by referring immediately to the cloudlet resources that are allocated in different amounts for different types of VMs.

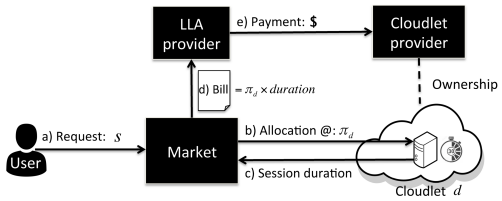


Fig. 2: FogSpot On-Demand Provisioning Overview

- A1) Each request is for a single VM.
- A2) Each request belonging to class p is associated with a bid, $b_{s,p}$ which expresses the user's willingness to pay for a VM per unit of engagement time to an LLA s instance.
- A3) Requests are not queued at the on-path markets, *i.e.,* the request is either served or rejected immediately.
- A4) The VM allocation time overhead has no impact on the LLA's QoS.

More specifically, let the price at cloudlet d be π_d. Upon the arrival of a request of class p for LLA s the market operates according to the following rules:

- R1 If there are no available VMs, reject the request.
- R2 Else,
  - If $b_{s,p} \geq \pi_d$, a VM of cloudlet d is allocated to serve the request at price π_d for each time unit of user's engagement.
  - Else if $b_{s,p} < \pi_d$, the request is rejected.

Assuming $b_{s,p} \geq \pi_d$ and available VMs exist, the allocation takes place and the cloudlet starts a timer for keeping track of the user's engagement duration (step b). After the session completion/application termination, the cloudlet informs the market about the engagement duration (step c). Then the market verifies the duration and converts it to a bill that equals the engagement duration times the agreed per time unit service price π_d (step d). Finally, the bill is sent to the corresponding LLA provider which eventually pays the cloudlet provider for its service.[3]

Next, we explain how bids $b_{s,p}$ for users' LLA requests are derived $\forall s \in \mathcal{S}$ and $\forall p \in \mathcal{P}$. Let $u^d_{s,p}$ be the per time unit QoS gain of class p LLA s when served at cloudlet d, instead of its default cloud. The truthfulness of a bid is defined in the following straightforward way:

Definition 1. *A bid $b_{s,p}$ for class p LLA s provisioning at cloudlet d is truthful iff $b_{s,p} := u^d_{s,p}$, where $u^d_{s,p}$ is the per time unit QoS gain of the requested LLA at d.*

In other words, a bid is truthful when it equals the actual gain of the served application.

Then, given a cloudlet d's price, π_d, the LLA s class p utility rate from using cloudlet d is defined as follows:

Definition 2. *The utility rate of class p LLA s from bid $b_{s,p}$ and QoS gain $u^d_{s,p}$ is:*

$$\text{utility rate} = \begin{cases} u^d_{s,p} - \pi_d, & \text{if } b_{s,p} \geq \pi_d, \\ 0, & \text{otherwise.} \end{cases}$$

[3]The technical details of the explained process, related to the security, verification, etc., are beyond the scope of this paper.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

Proposition 1. *Under rules R1-R2, the request bids are truthful.*

Proof. Let $b_{s,p}$ be the bid of a class p application s request with $u^d_{s,p}$. We investigate the following possible cases:

- If $u^d_{s,p} \geq \pi_d$ then the request would win a VM with a truthful bid as well as an overbid.
- If $u^d_{s,p} \geq \pi_d > b_{s,p}$ underbidding returns a utility rate equal to 0, as opposed to a truthful bid for which the utility rate is non-negative.
- If $\pi_d > u^d_{s,p}$ then the request would not allocate a VM with a truthful bid as well as an underbid.
- If $b_{s,p} > \pi_d > u^d_{s,p}$, then overbidding would return a negative utility rate, as opposed to a truthful bid for which the utility rate is 0.

From the previous cases it is clear that truthful bidding, *i.e.,* $b_{s,p} = u^d_{s,p}$, is the dominant strategy. □

Since LLA requests have to be truthful for a rational user, the strategy of their deployment is straightforward. In other words, each request has to be associated to its corresponding QoS per time unit gain at a given cloudlet *i.e.,* $b_{s,p} = u^d_{s,p}$ $\forall s \in \mathcal{S}$, $\forall p \in \mathcal{P}$, and $\forall d \in \mathcal{D}$.

1) Bid Derivation Practical Aspects: In practice, we envision that trusted software-based agents, residing at each cloudlet market, will derive and submit bids on behalf of user requests. An agent associates a request to a bid after estimating the QoS improvement (gain) that will be attained by the request's service at the current cloudlet as opposed to the default application cloud.

In the context of LLAs, the QoS gain is based on RTT measurements between *i*) the user, originating the request, and the cloudlet, and *ii*) the cloudlet and the default cloud of the requested LLA. Agents can straightforwardly measure RTT to the default cloud locations by periodically sending probe messages that trigger a reply at the other end. On the other hand, user-to-cloudlet measurements pose a scalability challenge due to the required per-user measurement state that has to be maintained at the agent's side; consequently, an efficient system would delegate this process to the user-side. In detail, similarly to user-side measurements currently deployed by major content distribution networks (CDNs) (among others [9]), the client side of a LLA could start sending on-path probe packets, that potentially interact with a set of cloudlets, as part of an initialisation process before sending the actual application request. The client LLA can piggyback the measurement results to each agent. In that way, agents obtain a measurement state for only the users whose requests might reach, if they will not be served earlier, to their market.

Finally, the agents perform the monetary valuation of the estimated QoS gain as dictated by the provider of each LLA. This is done in a straightforward way, *i.e.,* the bid equals the monetary valuation of the gain, due to the truthfulness property, *i.e.,* Proposition 1. Note that different classes requesting the same LLA result in different bids at the same cloudlet. Clearly, the closer a request is served to the cloud, the lower the QoS gain will be, and in turn, the lower the bid becomes. That is, at the cloud, bids and QoS gains become equal to zero.

B. FogSpot: Price Derivation Component

FogSpot's price derivation component associates each cloudlet $d \in \mathcal{D}$ to a price, π_d, by deploying a Dutch auction mechanism which is particular useful for the uncoordinated execution of FogSpot as we analyse in Section V. We start by presenting FogSpot's underlying queueing model of each cloudlet, dictated by the on-demand provisioning component and the Poisson process LLA requests generation assumption that create the basis of price derivation described next. We conclude by discussing FogSpot's truthfulness concerns at the stage of price derivation.

1) FogSpot Cloudlet Queueing Model: FogSpot on-demand provisioning component treats each cloudlet d as a First Come Fist Served (FCFS) Queueing system of C_d identical parallel servers with zero-sized queues, since the requests are either accepted or rejected immediately without queueing. A cloudlet receives requests at exponential interarrival times with rate $\sum_{s\in\mathcal{S}}\sum_{p\in\mathcal{P}_d} y^d_{s,p}$ while the user engagement duration, *i.e.,* the service time of a request, has application-specific exponential interarrival times of rate μ_s $\forall s \in \mathcal{S}$. In other words, each cloudlet d is a M/M/C_d/C_d/FCFS system, dictating the admitted provisioning rate for LLA s at d, $Y^d_s = \sum_{p\in\mathcal{P}_d} y^d_{s,p}$, as we explain next.

Let X^d_s be the arrival rate of LLA s requests that arrive at cloudlet d. The effective rate at which LLA s can be provisioned, Y^d_s, equals:

$$\begin{aligned} Y^d_s &= X^d_s \times P_d(\text{Cloudlet is not fully occupied}) \\ &= X^d_s \times \Big(1 - P_d(\text{Cloudlet is fully occupied})\Big) \\ &= X^d_s \times \Big(1 - P_d(n = C_d)\Big). \end{aligned} \tag{4}$$

Clearly, requests of LLA s are rejected from cloudlet d at rate $X^d_s - Y^d_s$, while the steady state probability of having all VMs occupied, $P_d(n = C_d)$, is:

$$\begin{aligned} P_d(n = C_d) &= \frac{\rho^{C_d}}{C_d!} P_d(n = 0) \\ &= \frac{\rho^{C_d}}{C_d!} \left[\sum_{n=0}^{C_d} \frac{\rho^n}{n!}\right]^{-1}, \end{aligned} \tag{5}$$

where parameter ρ is $\rho = \sum_{s\in\mathcal{S}} X^d_s/\mu_s$, as described in [19]. Note that rate Y^d_s characterises a Poisson process. Lastly, it can be shown that, as we increase the number of VMs at d, *i.e.,* capacity C_d, $Y^d_s \to X^d_s$.

2) Price Derivation Process: Next we describe the steps of the price derivation component when applied to cloudlet d, resulting in price π_d, assuming that the remaining cloudlets in the network have already derived their prices. We start by initialising price π^t_d with a very high value $\pi_{\max}$, *i.e.,* a few times higher than any reasonable market price, so that $Y^d_s(t) = 0$ $\forall s \in \mathcal{S}$ at $t = 0$. Then the steps followed are:

Step 1: Each LLA provider s estimates the traffic that could be served at price π^t_d, $X^d_s(t)$. At first each LLA provider s identifies the classes whose gain after the price reduction is non-negative when served at d, $\mathcal{P}_{d,s}(\pi^t_d) \subseteq \mathcal{P}_d$, defined as:

$$\mathcal{P}_{d,s}(\pi^t_d) = \Big\{p : d \in \mathcal{D}_p \text{ and } u^d_{s,p} \geq \pi^t_d\Big\}. \tag{6}$$

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

Then:

$$X_s^d(t) = \sum_{p \in \mathcal{P}_{d,s}(\pi_d^t)} \Big(\lambda_{s,p} - \sum_{d' \in \mathcal{D}_p : u_{s,p}^{d'} > u_{s,p}^{d}} y_{s,p}^{d'} \Big), \quad (7)$$

where $\sum_{d' \in \mathcal{D}_p : u_{s,p}^{d'} > u_{s,p}^{d}} y_{s,p}^{d'}$ is the request provisioning rate of LLA s over the cloudlets with a higher QoS with respect to class p, *i.e.,* the cloudlets that are deployed between the end users and cloudlet d along a path that defines class p and therefore are *closer* to the end users. The LLA s then submits value $X_s^d(t)$ to cloudlet d. Note that cloudlet d does not require information related to requests' classes, since it is interested in the aggregate rate of each LLA s, $X_s^d(t)$.

Step 2: Cloudlet d estimates the provisioning request rate of LLA s, $Y_s^d(t)$, from (4). Note that the information about cloudlet's capacity in terms of VMs, C_d, does not have to be shared with the LLA providers, who are only interested in the request provisioning rate of their service at d.

Step 3: Cloudlet d applies one of the following termination criteria:

Revenue maximisation (RevM): The Dutch auction terminates if $\pi_d^t = 0$. Then, the price is set based on the candidate price that maximises the revenue of cloudlet d :

$$t^* = \arg\max_t \Big(\pi_d^t \times \sum_{s \in \mathcal{S}} Y_s^d(t) \frac{1}{\mu_s} \Big). \quad (8)$$

Then $\pi_d = \pi_d^{t^*}$ and cloudlet d serves LLA s at rate $Y_s^d = Y_s^d(t^*)$.

Social welfare maximisation (SWM): The Dutch auction terminates either when the provisioning requests cover the cloudlet's capacity, *i.e.,* $\sum_{s \in \mathcal{S}} Y_s^d(t)/\mu_s \geq \rho_{\text{thres}} C_d$, or when $\pi_d^t = 0$, in both cases SWM policy serves the highest possible volume of requests. The price is set to $\pi_d = \pi_d^t$ while the accepted request rates are $Y_s^d = Y_s^d(t)\ \forall s \in \mathcal{S}$.[4]

Step 4: Cloudlet d decreases price by $\Delta\pi$ according to the Dutch auctions, *i.e.,* $\pi_d^{t+1} = \pi_d^t - \Delta\pi$, and proceeds to the next iteration, $t = t + 1$, by returning to Step 1.[5]

3) Addressing Truthfulness Concerns: Assume that LLA s is untruthful in the provisioning request rate that declares at cloudlet d, $\tilde{X}_s^d$ instead of the truthful rate X_s^d. Then if $\tilde{X}_s^d > X_s^d$ cloudlet d will eventually detect the false declaration by observing the under-utilisation of its resources by LLA s. Similarly, if LLA s declares a $\tilde{X}_s^d < X_s^d$ the derived price at d will be reduced from the ideal price π_d to $\tilde{\pi}_d$, *i.e.,* $\tilde{\pi}_d < \pi_d$, which means that the actual admitted requests will be higher than the expected ones. Then cloudlet d will again be able to detect the false declaration by observing the over-utilisation of its resources by LLA s. Especially when cloudlet d applies a SWM approach, a price $\tilde{\pi}_d < \pi_d$ increases the competition for resources and LLA s will have less chances in finding available VMs since cloudlets aim the highest possible utilisation of their resources.

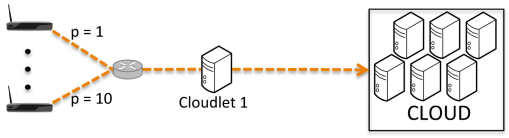


Fig. 3: Toy Topology, Single cloudlet receiving requests from 10 classes.

C. Detailed Scenario

Next we present step-by-step the execution of FogSpot when applied over a toy topology with a single cloudlet, *i.e.,* Cloudlet 1, that serves 10 classes of requests as depicted in Fig. 3. We consider a single LLA s whose request classes experience the QoS gains of $\{37.15, 41.55, 47.03, 53.3, 60.25, 67.87, 76.15, 85.11, 94.83, 100.0\}$ upon getting served at Cloudlet 1. Furthermore, the average engagement duration of a user to an LLA instance is set to 60 sec, while each request has an equal probability of being generated by any Class and at the total rate of 1, *i.e.,* $\sum_{p \in \{1,2,..,10\}} \lambda_{s,p} = 1$. Lastly, we consider that Cloudlet 1 can support up to 20 VMs.

1) Price Derivation Component Execution: In the beginning, FogSpot applies its price derivation component to associate Cloudlet 1 with a price. Without loss of generality, we assume a linear relationship between the QoS gain and the amount of money that a user is willing to pay, *i.e.,* a user will pay X$ per second of engagement for every single unit of QoS improvement. That is, the maximum price that a request could possible pay is 100 X$ per second of engagement that defines the price at the beginning of the price derivation execution, *i.e.,* $\pi_{\max} = 100$. Lastly, we set the decremental price step of Dutch auction, $\Delta\pi$, equal to 0.05, and the utilisation target level, ρ_{thres}, equal to 0.93.

The step-by-step execution of the price derivation component is presented in Table II. Specifically, the first column indicates the iteration of execution, the second the price at this iteration, the third the arrival rate of LLA requests that can be served at Cloudlet 1, the fourth the effective arrival rate of requests that is actually served due to the zero queue size, the fifth the expected number of utilised VMs, and the last one the expected revenue (in X$) per second for Cloudlet 1. As the price decreases, according to the applied Dutch auction, both the effective arrival rate of requests (column 4) and expected utilised VMs (column 5) increase. Table II includes only the iterations where requests' arrival rate (column 3) grows as response to Dutch auction's price reduction. Cloudlet 1 achieves the SWM target utilisation of its resources upon iteration 796, *i.e.,* $18.9853 > 0.93 \times 20$, while the maximum revenue is achieved at iteration 298 and 85.1 X$ price. That is, the price of Cloudlet 1 is set to 60.2 if SWM policy is applied and to 85.1 in the case of RevM policy. After setting the price the upcoming requests interact with Cloudlet 1 according to FogSpot's on-demand provisioning component as we show next. Note that Cloudlet 1 is never completely utilised even for a price equal to 0, as we see in iteration 2000. This is because in the case of zero queue size, a request can never

[4]Note that, ρ_{thresh} serves the purpose of achieving a price higher than 0, since from Eq. 4 we see that the utilisation of the system can never be 100%.

[5]$\Delta\pi$ can be chosen so that $\pi_d^t \geq 0$ always holds.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

TABLE II: FogSpot Price Derivation Detailed Execution

t	π_d^t	$X_s^d(t)$	$Y_s^d(t)$	$Y_s^d(t)/\mu_s$	$\pi_d^t Y_s^d(t)/\mu_s$
0	100	0.1	0.0999	5.9999	599.9977
104	94.8	0.2	0.1980	11.8824	1126.4564
298	**85.1**	0.3	0.2672	16.0341	**1364.5077**
478	76.1	0.4	0.2971	17.8300	1356.8644
643	67.85	0.5	0.3099	18.5974	1261.8372
796	**60.2**	0.6	0.3164	**18.9853**	1142.9194
935	53.25	0.7	0.3201	19.2115	1023.0166
1060	47.0	0.8	0.3226	19.3576	909.8079
1170	41.5	0.9	0.3243	19.4589	807.5475
1258	37.1	1.0	0.3255	19.5331	724.6804
2000	0.0	1.0	0.3255	19.5331	0.0

arrive at a cloudlet at the exact moment when a VM becomes available. That is, SWM policy is able to achieve a positive spot price, despite the fact that it aims for the highest possible utilisation of a cloudlet's resources, due to the target utilisation parameter ρ_{thres}.

2) On-Demand Provisioning Component: As requests of LLA s arrive at Cloudlet 1, on-demand provisioning component accepts a subset of them based on the derived price and VM availability at the time of request arrivals. Fig 4a depicts the accepted and rejected requests by Cloudlet 1, over 600 seconds of simulation[6], when SWM policy is applied. In this case, only requests with at least 60.2 QoS units gain are accepted in accordance to the price assigned to Cloudlet 1 by the price derivation component.

RevM policy similarly serves requests with at least 85.1 QoS units gain. That is, when comparing SWM to RevM policy there are more requests accepted by the first that leads to a higher utilisation of VMs as captured by their moving VM utilisation average in Fig. 4b. On the other hand, SWM results in a lower revenue per second compared to RevM, as depicted in Fig. 4c. Note that both moving averages of utilisation and revenue converges to the estimated values of Table II.

V. FogSpot Game Theoretic Analysis

In this section, we analyse the uncoordinated price derivation performance of FogSpot mechanism, *i.e.,* each cloudlet is assigned a price periodically in unsynchronised way, from a game theoretic perspective.

A. FogSpot Cloudlet Pricing as a Game.

We consider each cloudlet d as a player whose action is related to the price selection over a set of prices $\Pi_d = (u_{s,p}^d : \forall s \in \mathcal{S}, \forall p \in \mathcal{P}_d)$. We associate each cloudlet d to a payoff function $\mathcal{R}_d(\cdot)$ mapping $\Pi_1 \times \Pi_2 \times ... \times \Pi_D$ to a real value, where the elements of $\Pi_1 \times \Pi_2 \times ... \times \Pi_D$ are referred as *action combinations* or *states*. That is, *pure Nash equilibrium* is defined as a state $\pi = (\pi_1, ..., \pi_D)$ such that for each cloudlet d there is no action that can increase its payoff, *i.e.,* $\mathcal{R}_d(\pi_1, .., \pi_d, ..., \pi_D) \geq \mathcal{R}_d(\pi_1, .., \pi'_d, ..., \pi_D)$ for any $\pi'_d \in \Pi_d$.

Consider a directed graph $\mathcal{G} = (\Pi, E)$ which nodes consist of the states in $\Pi = \Pi_1 \times \Pi_2 \times ... \times \Pi_D$ space and edges E connect only states that differ by a single component causing an improvement to the corresponding player's payoff function, *i.e.,* there is an edge (π, π') iff states $\pi = (\pi_1, .., \pi_d, ..., \pi_D)$ and $\pi' = (\pi_1, .., \pi'_d, ..., \pi_D)$ differ only in component π'_d and $\mathcal{R}_d(\pi') > \mathcal{R}_d(\pi)$. From [23] we know that if the graph is acyclic then the *Nash dynamics converges* to a pure Nash equilibrium (the sinks of the graph), leading us to the proposition presented in [15] :

Proposition 2. *If the Nash dynamics converges, then there is a pure Nash equilibrium.*

Next we investigate the settings under which FogSpot derives a pure Nash equilibrium to the game of cloudlet pricing for stable LLA demand conditions, *i.e.,* $\lambda_{s,p}$ remains stable for each $p \in \mathcal{P}$ and $s \in \mathcal{S}$. Nash equilibrium indicates that FogSpot derives a stable solution in an uncoordinated way.

B. FogSpot Nash Equilibrium Derivation of SWM Approach

Let $Y_s^d(\pi)$ be the provisioning rate of service s at cloudlet d when the state of prices is π. The payoff function of SWM policy in FogSpot is:

$$\mathcal{R}_d^{\text{SWM}}(\pi) = \begin{cases} 1, & \text{if } \sum_{s \in \mathcal{S}} Y_s^d(\pi)/\mu_s \geq \rho_{\text{thres}} C_d \\ & \text{or } \pi_d = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Note that as π_d decreases, $\sum_{s \in \mathcal{S}} Y_s^d(\pi)/\mu_s$ increases.

Theorem 1. *FogSpot mechanism converges to a pure Nash equilibrium with the highest possible payoff, in the pricing game, when the SWM policy is applied.*

Proof. Let the current state of prices be $\pi = (\pi_1, ..., \pi_D)$ and cloudlet d is the next that triggers the price derivation mechanism. Then we know for the new price of cloudlet d, π'_d, derived by the SWM policy will be either *i*) $\pi'_d < \pi_d$, in case that $\pi_d > 0$ and $\mathcal{R}_d^{\text{SWM}}(\pi) = 0$ the new price will either comply to $\sum_{s \in \mathcal{S}} Y_s^d(\pi')/\mu_s \geq \rho_{\text{thres}} C_d$ or $\pi'_d = 0$, *i.e.,* $\mathcal{R}_d^{\text{SWM}}(\pi') = 1$, otherwise *ii*) $\pi'_d = \pi_d$ and therefore state $\pi' = \pi$. Clearly, FogSpot's SWM approach's execution forms a directed graph whose nodes cover the states existing in space Π. SWM policy starts from state $\pi_{\text{init}} = (\pi_1 = \pi_{\max}, ..., \pi_D = \pi_{\max})$, *i.e.,* $Y_s^d(\pi_{\text{init}}) = 0 \ \forall d \in \mathcal{D}$ and $\forall s \in \mathcal{S}$. Then the execution of the mechanism by a cloudlet d leads to a new state $\pi = (\pi_1 = \pi_{\max}, ..., \pi_d, ..., \pi_D = \pi_{\max})$ where $\pi_d < \pi_{\max}$ and $\mathcal{R}_d^{\text{SWM}}(\pi) > \mathcal{R}_d^{\text{SWM}}(\pi_{\text{init}})$. In general, edges in SWM policy graph connect only states that differ by a single price that is always decreasing, causing an improvement to the payoff function of the corresponding cloudlet. Clearly, this graph is acyclic since under no circumstances a price is increasing and therefore the Nash dynamics converges to a pure Nash equilibrium. □

C. FogSpot Nash Equilibrium Derivation of RevM Approach

Next we focus on the RevM policy, whose payoff function is:

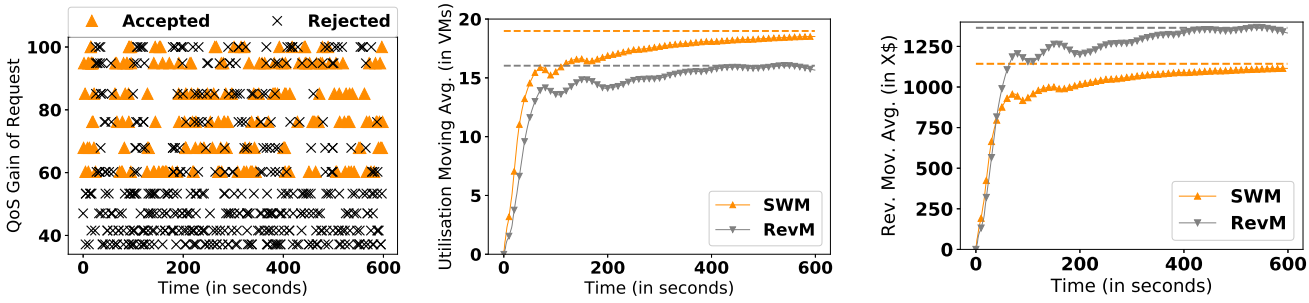
$$\mathcal{R}_d^{\text{RevM}}(\pi) = \pi_d \sum_{s \in \mathcal{S}} Y_s^d(\pi)/\mu_s. \tag{10}$$

Unfortunately, there are no guarantees that given a state π the execution of the RevM policy in an uncoordinated way will either decrease or increase the price of Cloudlet 1. Hence, we

[6]The details are provided in Section VI.

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing



(a) SWM policy accepted/rejected requests over time.

(b) Policies comparison, VMs utilisation moving average over time.

(c) Policies comparison, revenue moving average over time.

Fig. 4: FogSpot, Detailed Scenario, On-Demand Provisioning Component

focus on the special case of fog topologies where cloudlets are deployed in a *hierarchical* way defined as:

Definition 3. *Two cloudlets are associated with a hierarchical relationship if the actions of the first do not affect the payoff of the second.*

In other words, a cloudlet d is hierarchically lower than a cloudlet d' if there is no class and LLA such that $u^{d}_{s,p} > u^{d'}_{s,p}$ while the relationship $u^{d}_{s,p} < u^{d'}_{s,p}$ is true for at least one class and LLA. In a sense, cloudlet d' is located closer to an end user meaning that by setting its price it determines the rate of LLA requests that serves; consequently affecting the rate of requests left to get forwarded to cloudlet d.

Two cloudlets can also be uncorrelated as we define next:

Definition 4. *Two cloudlets are considered uncorrelated if their actions do not affect the payoff of each other.*

In detail, a cloudlet d is uncorrelated to a cloudlet d' if there is no class for any LLA that traverses both of them, *i.e.,* $\mathcal{P}_d \cap \mathcal{P}_{d'} = \{\emptyset\}$. Hence, the rates of provisioning each LLA to one of the two cloudlets, does not affect the provisioning rates at the other.

Definition 5. *A fog infrastructure is considered hierarchical if all the deployed cloudlets are associated either by a hierarchical or uncorrelated relationship.*

Theorem 2. *The FogSpot RevM approach converges to a pure Nash equilibrium, in the pricing game when applied in hierarchical fog topologies.*

Proof. Let the current state of prices be $\pi = (\pi_1, ..., \pi_D)$ and the RevM policy is applied to cloudlet d next. Without loss of generality assume that cloudlet indexes imply the hierarchy of the topology, *i.e.,* only cloudlets 1 to $d-1$ have a hierarchical relationship with cloudlet d, meaning that their price affects the payoff of d. Then if $d = 1$ the first execution of RevM policy will derive cloudlet's optimal price π^*_1 that is not affected by any other cloudlet price in the fog, *i.e.,* $\mathcal{R}^{\text{RevM}}_1(\pi^*_1, ..., \pi_D) \geq \mathcal{R}^{\text{RevM}}_1(\pi)$ for all $\pi \in \Pi$. In general, if prices π_1 to π_{d-1} are optimal, for $d > 1$, then the newly derived price of d, π^*_d, is optimal in the sense that maximises its expected revenue, since cloudlets 1 to $d-1$ will not change their price in the future, *i.e.,* $\mathcal{R}^{\text{RevM}}_{d'}(\pi^*_1, ..., \pi^*_d, ..., \pi_D) \geq \mathcal{R}^{\text{RevM}}_{d'}(\pi)$ for each $d' \in [1, 2, .., d]$ and all $\pi \in \Pi$. □

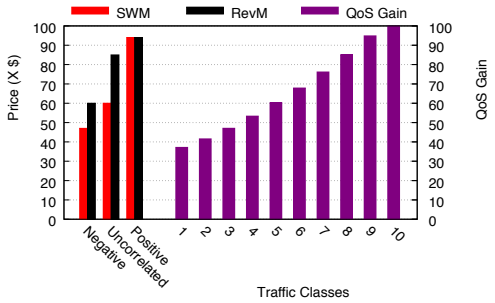


Fig. 5: Toy Setting, FogSpot Price for different demand-class QoS Gain correlations

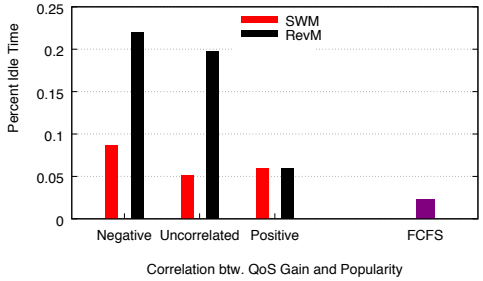


Fig. 6: Toy Setting, Idle Time Percentage of Resources for different demand-class QoS Gain correlation vs. FCFS.

VI. Performance Evaluation

In this section, we demonstrate FogSpot's performance. We begin by revisiting the toy setting of the detailed scenario, presented in Section IV-C, for performing additional small scale experiments that shed light on FogSpot's behaviour, before proceeding to simulation results in realistic topologies. We implement FogSpot and the baseline comparison approaches, described next, by extending ICARUS caching simulator for application provisioning problems [24].

A. FogSpot Toy Setting

1) QoS Gain-Demand Correlation Experiments: Similar to the detailed scenario of Section IV-C, we assign a capacity

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

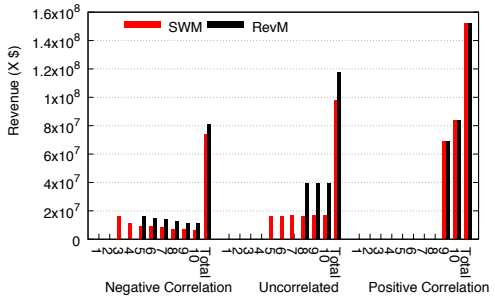


Fig. 7: Toy Setting, Per Class Revenue for different demand-class QoS Gain correlation.

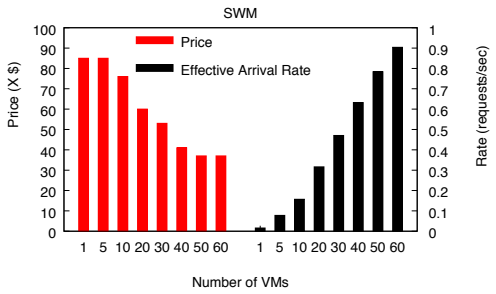


Fig. 8: Toy Setting, SWM policy: Price vs. Effective Rate for an increasing number of VMs

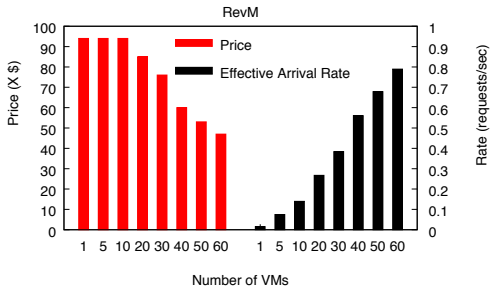


Fig. 9: Toy Setting, RevM policy: Price vs. Effective Rate for an increasing number of VMs

of 20 VMs to Cloudlet 1 of Fig. 3, and we investigate the behaviour of FogSpot in the following scenarios:

- *Positive correlation of demand and QoS gain* where the most popular (*i.e.,* generating the most requests) class is Class 10 with QoS gain equal to 100, the second most popular is Class 9, *etc.*
- *Negative correlation of demand and QoS gain* where the most popular class is Class 1 with QoS gain equal to 37.15, the second most popular is Class 2, *etc.*
- *Uncorrelated demand and QoS gain* where requests have equal probability of falling into any class.

The total request generation rate is equal to 1 while the request correlation is expressed via a Zipf distribution of exponent 0.8 that categorises each request to one of the 10 classes. Specifically, the highest probability is assigned to Class 10 (Class 1) in the case of the positive (negative) correlation.

In Fig. 5 we plot the price (in X$ per QoS gain unit and second of engagement[7]) achieved by FogSpot for the 3 demand correlation scenarios. The RevM policy derives identical prices to SWM policy for the case of positive correlation since the requests of classes 9 and 10 are sufficient for utilising the cloudlet resources, *i.e.,* the price is set equal to the QoS gain of class 9. That said, the RevM policy in the cases of uncorrelated and negatively correlated demand has the opportunity of trading utilisation/idle time for increasing the cloudlet's revenue. In particular, in the advent of negative correlation (no correlation) the RevM policy sets a price that serves classes 5 to 10 (8 to 10), while SWM focuses on achieving a satisfying utilisation of the available VMs by accepting to serve classes 3 to 10 (5 to 10).

The tradeoff between resource utilisation and revenue increase is depicted in Figs. 6 and 7. In Fig. 6, the percentage of time VMs remain idle in RevM policy is 3 times (5 times) bigger than the corresponding SWM idle time of 8% (5%) for the case of negative correlation (no correlation). However, the idle percentage time in all cases and policies remains higher than the baseline FCFS policy that serves all classes, *i.e.,* in FCFS the cloudlet's price is set to zero. Lastly, the idle percentage time overhead of RevM policy comes with the merits of a higher revenue as we observe in Fig. 7.

2) Cloudlet Capacity Impact Experiments: Next we focus on the case of uncorrelated arriving requests, *i.e.,* equal probability of request generation for each class, and we illustrate FogSpot performance as we increase the number of VMs at Cloudlet 1 of the toy setting topology. Figures 8 and 9 show the impact of VM quantity on the price and the effective rate of SWM and RevM policies. Clearly, the effective/admitted rate of SWM policy is strictly higher than that of RevM, since in RevM each cloudlet trades its resource utilisation for a greater price that leads to a higher revenue as we see in Fig. 10.

[7]Assuming a linear relationship between prices and QoS gains.

B. Simulation Results

Next we focus on FogSpot performance on realistic topologies.

Application categories for LLAs: The QoS of the LLAs is expressed as a decreasing function of the end user perceived latency to each cloudlet [11], [32]. We consider that each LLA is characterised by a decreasing QoS function of the latency having the general form:

$$u(x) = \left(\frac{u_{\min}}{u_{\max}} + \left(1 - \frac{u_{\min}}{u_{\max}}\right) \left(1 - \frac{x - l_{\min}}{l_{\max}}\right)^{\frac{1}{\alpha}} \right) \times u_{\max} \quad (11)$$

The constants u_{max} ($u_{\min}$) represents the maximum (minimum) QoS that the application user can achieve at the minimum (maximum) latency $l_{\min}$ ($l_{\max}$), *i.e.,* $u(l_{\min}) = u_{\max}$ and $u(l_{\max}) = u_{\min}$. We set $u_{\max} = 100$, $l_{\min} = 5$ ms[8] when it is not specified differently. Moreover, function $u(\cdot)$ is convex for $0 < \alpha \leq 1$; that is, we set $\alpha = 0.2$ since LLAs' QoS is expected to be more sensitive to latency changes closer to $l_{\min}$. We depict the QoS function for α values 0.2 and 1.0 in Fig. 11.

[8]With recent advances in LTE technology, mobile operators reported handset-to-base-station latencies around 2 msec (RTT of 5 ms), see: http://news.itu.int/with-5g-looming-sk-telecom-reduces-lte-latency-to-just-2ms

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

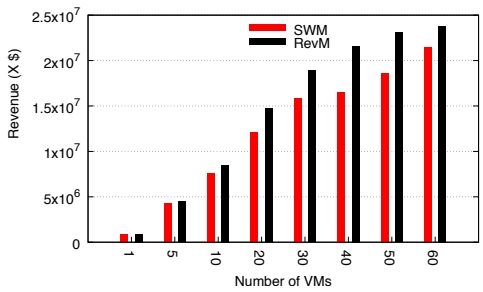


Fig. 10: Toy Setting, Revenue for different number of VMs.

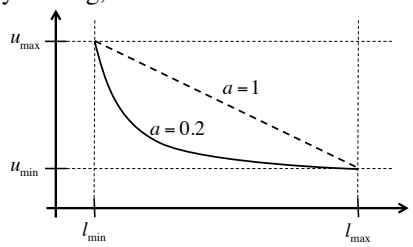


Fig. 11: The QoS function we consider for each LLA category.

Based on Eq. 11, we create ten *LLA categories*, each associated to a QoS function that models a certain sensitivity of the LLA to network latencies. We assign to the ten LLA categories a unique u_{min} value from the set: $\{0, 10, 20, \ldots, 90\}$. In this way, different application categories have different gains from being provisioned at a cloudlet, varying in the best case from 10 for $u_{\min} = 90$ (less latency sensitive LLAs) to 100 for $u_{\min} = 0$ (latency critical LLAs), when provisioned at the edge. We consider the number of ten LLA categories sufficient for the purpose of our evaluation since it is comparable to the currently considered types in the context of IoT [7], [21] and Tactile Internet [16]. We assume that users remain engaged to their LLA instance on average for 1 minute while each user selects one of the ten LLA categories with equal probability, *i.e.,* 10%. We execute the following experiments for the duration of three hours.

Topologies: We evaluate FogSpot under:

- *A Small Tree topology*: We consider the setting of Fig. 12 that depicts a small binary tree of seven cloudlets placed over 3 levels that create a round-trip-time to the cloud equal to $l_{\max} = 300$ ms. At each of the 3 leaf nodes, we generate one request per second.
- *An ISP topology*: We use the Tiscali topology from the Rocketfuel dataset [27] that contains 240 nodes and 404 links. Among the 240 nodes, we designate as hosts the 79 nodes that present a degree of one, *i.e.,* they are connected to only a single node of the topology. After that, we place the cloud at the node with the highest closeness centrality with respect to all hosts, *i.e.,* shortest average distance to all hosts. Then, we set each host to generate one request per second forwarded to the cloud via the shortest path in terms of latency. We place a cloudlet to each node of the topology that belongs to at least a single shortest path from a host to the cloud. As a result, there are 80 cloudlets in the topology for reducing the cloud's RTT latency that on average is accessible at 155 ms from each host.

Comparison approaches: We compare RevM and SWM

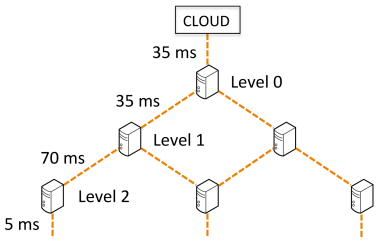


Fig. 12: Small Tree Topology.

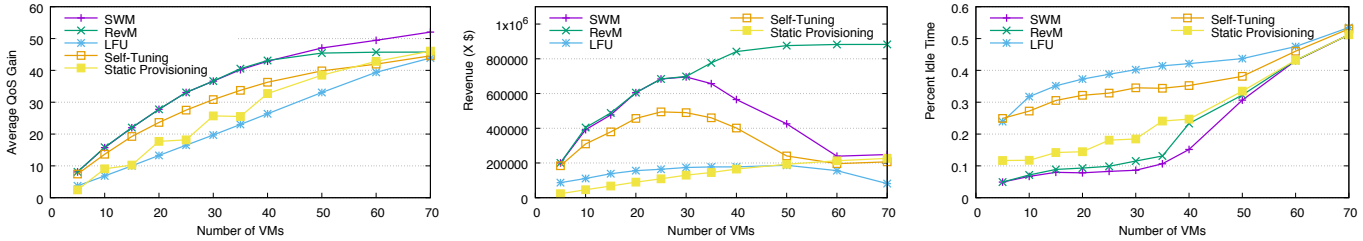
policies of FogSpot against the following approaches:

- ***Static Provisioning*:** Cloudlets assign their VMs to application instances according to their demand at once, *i.e.,* an equal number of VMs is assigned to each LLA due to the uniform request distribution we assume.
- ***Least Frequently Used (LFU) provisioning*:** Cloudlets periodically assign their VMs to LLA instances, prioritising the ones with the highest observed demand, *i.e.,* popularity. In this case, VMs are allocated first to the most popular application, and the process continues with second most popular *etc.* , until all the VMs of a cloudlet are allocated. The number of VMs assigned to each application is determined according to the number of both the accepted and rejected requests by a cloudlet.
- ***Self-Tuning Provisioning*:** Cloudlets periodically assign their VMs to LLA instances by prioritising the ones with the highest QoS gain. In particular, this strategy iterates through the applications, sorted in decreasing order of QoS gain, and assigns the corresponding number of unallocated VMs. This results with VMs being assigned to applications that are willing to *pay the most*. The number of VMs assigned to each application depends once again on its observed demand.

The Static approach is a form of *proactive* provisioning, since it relies on prior knowledge of the demand (*i.e.,* request volume) for each application, while LFU and Self-tuning approaches perform *reactive* provisioning, as each cloudlet periodically reassigns their VM instances to individual LLAs based on the request patterns received during the most recent observation period. We set the length of the observation period to 6 minutes. Note that we consider a single LLA per LLA category in order to give an advantage to the above-mentioned approaches against FogSpot's on-demand provisioning, otherwise their performance could be arbitrarily deteriorated. For example, by increasing the number of LLAs per category from 1 to 10, the utilisation of each cloudlet would decrease to the 10% of the currently achieved utilisation for the Static, Self-tuning, and LFU provisioning with a negative impact on the achieved QoS gain as well as cloudlets' revenue. Lastly, in order to perform a straightforward and fair comparison against FogSpot, we assume that the comparison approaches leverage a pay-as-you-go charging mechanism, identical to FogSpot, where each cloudlet is associated to a price that equals the lowest QoS gain over all the accepted LLA request classes at a given location. In that way, we avoid creating application specific prices that challenge requests' bidding truthfulness since they would pay different prices for identical VMs.

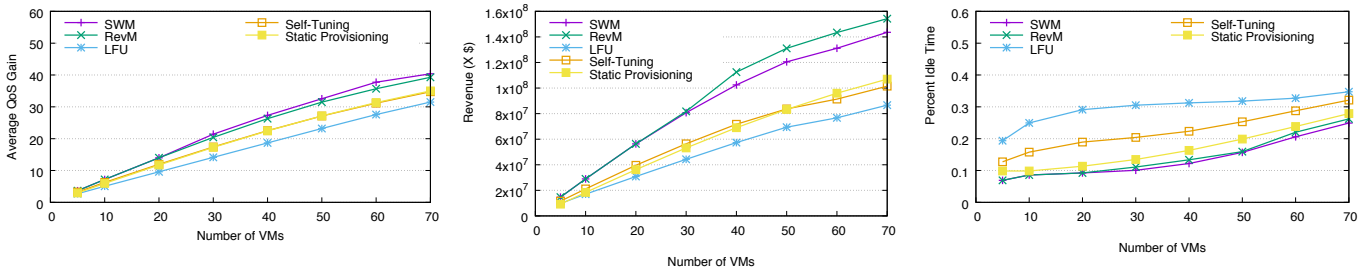
1) Cloudlet Capacity Impact Experiments: In this experiment, we set $\rho_{\text{thres}} = 0.9$ and increase the number of VMs

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing



(a) QoS Comparison (b) Revenue Comparison (c) Idle Time Percentage Comparison

Fig. 13: Small Tree, Number of VMs impact.



(a) QoS Comparison (b) Revenue Comparison (c) Idle Time Percentage Comparison

Fig. 14: ISP, Number of VMs impact.

that are supported by each cloudlet for both topologies[9]. In Fig. 13, we plot the QoS gain, Revenue, and Idle Time percentage for various per-cloudlet VM quantities. Fig. 13a depicts the QoS gain for the described approaches for the case of small tree topology. Clearly, the QoS gain is an increasing function of VM quantity for all strategies, *i.e.,* the more VMs the better, with FogSpot SWM and RevM consistently achieving better QoS than the others by more than 10%. Self-tuning approach comes next followed by Static provisioning, which interestingly outperforms LFU. The reason is that under the uniform request distribution assumption, it is better to statically assign VMs according to their distribution instead of attempting to host the currently most popular LLAs, as in the case of LFU. Note that the performance of the comparison approaches to FogSpot would be much worse if we consider more LLAs per category.

Furthermore, SWM and RevM FogSpot policies have identical performance for up to 30 VMs per cloudlet. After that, RevM starts trading the utilisation of cloudlet's VMs for increasing revenue (Fig. 13b) as opposed to SWM that aims the full utilisation of VM resources, *i.e.,* the minimum possible Idle Time Percentage (Fig. 13c). In particular, RevM continues increasing its revenue, in terms of X$, while the revenue of SWM expectedly decreases. Specifically, for 70 VMs the RevM policy has 4× higher revenue than the other approaches. Regarding the remaining policies, Static and LFU present the lowest revenues, since they do not consider the QoS gain for their LLA provisioning, while Self-tuning and LFU demonstrate the highest idle times due to their reactive provisioning nature that does not take into account the requests' distribution.

[9]The exact value of ρ_{thres} is setting specific, and in our default evaluation setting the value of 0.9 is chosen based on the performance diversification of SWM and RevM policies, *i.e.,* a low ρ_{thres} would force the SWM policy to increase its idle time and eventually behave like the RevM one.

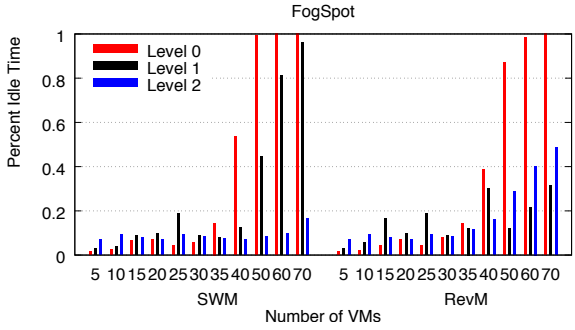


Fig. 15: Small Tree, Number of VMs impact, Average Idle Time Percentage Per Layer, FogSpot SWM vs. RevM.

A more detailed picture of how the idle time is propagated to the higher levels of the topology is depicted in Fig. 15. Each level has similar idle time for up to 35 VMs for both SWM and RevM. After that, in RevM approach the idle time starts increasing at Level 2, *i.e.,* leaf nodes, affecting the higher levels of the tree which in comparison to the SWM approach present lower idle times, since more requests are rejected from Level 2 and get forwarded towards the cloud.

In Fig. 14, we see similar trends for the default ISP topology with FogSpot outperforming again the comparison approaches.

2) Impact of Cloudlet Resource Allocation Strategies: In this section, we consider two resource allocation strategies that determine how computation resources are assigned to nodes in the network. First, we present an *incremental deployment* scenario, where an increasing number of cloudlets with identical resources are assigned one-by-one to a subset of the nodes in the ISP topology. In the second scenario, we consider a *non-uniform allocation of resources*, where cloudlets with a varying amount of resources are deployed over the ISP topology subject to a total budget of VMs. We

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

discuss the two allocation strategies and evaluate their impact on the performance below.

Incremental Deployment: We consider a scenario where identical cloudlets, in terms of VM capacities, are placed at individual nodes starting from the most central ones. A node's centrality is computed according to shortest-path betweenness centrality metric; however, in the centrality calculation we include only the shortest paths between the hosts and the cloud, as opposed to all node pairs of the topology. As a result, this allocation strategy deploys resources starting from the nodes with the highest concentration of arriving requests. In this scenario, we assume that each cloudlet has 40 VMs, which is roughly the resource amount where SWM and RevM policies diverge in terms of revenue as shown in Fig. 14b.

Fig. 16 depicts the performance of the strategies over a range of 10 to 80 deployed cloudlets, with 80 indicating their full deployment in the default topology. We observe that SWM achieves the highest QoS among all the strategies while as more cloudlets are added to the topology, the QoS gap between SWM and the rest of strategies marginally increases. Expectedly, SWM also achieves the lowest overall idle time (Fig. 16c) among all the strategies, while RevM achieves the highest revenue (Fig. 16b) as observed in the previous experiments. Furthermore, the idle time is roughly constant for the RevM strategy as opposed to an upwards trend presented by the rest of the strategies when resources are incrementally deployed in a top-down fashion, *i.e.,* starting from nodes near the cloud and expanding towards the edges. We observed a different behaviour with the bottom-up incremental deployment (using the reciprocal of centrality to select nodes) of cloudlets with idle times decreasing as the number of cloudlets increases due to lower concentration of arriving requests at the edges; however, we omit those results due to space limitations.

Non-uniform Resource Allocation: In this scenario, given a total computation budget in terms of number of VMs, we distribute the budget onto individual nodes (along shortest paths from hosts to the cloud) corresponding to cloudlet locations in the topology. We consider two ISP topologies in addition to the default Tiscali: AT&T and Exodus from Rocketfuel dataset [27]. AT&T (Exodus) topology is considerably larger (smaller) than Tiscali with 216 (33) nodes and 248 (31) hosts. Each host in all topologies generates approximately one request per second.

Each node obtains a share of the total VMs' budget proportional to its betweenness centrality, *i.e.,* cloudlets located in central nodes support more VMs, where the centrality is estimated as in the incremental deployment allocation setting. For each node, we obtain a number of VMs by probabilistically rounding up or down the ratio of node's centrality divided by the total centrality of all nodes in the network. In general few nodes present a high centrality compared to the rest in each ISP topology. Specifically, the Cumulative Distribution Function (CDF) of VMs' budget over the nodes assigned, sorted in increasing centrality order, follows a long-tail distribution where the ten most central nodes obtain approximately the 53%, 51%, and 46% of the resources budget in Exodus, Tiscali, and AT&T topologies, respectively. We omit the CDF plots due to space limitations.

We set the total VM budget in each ISP topology equal to the product of the number of hosts and expected engagement time (in secs) of requests. The reason is that we want to deploy a sufficient amount of VMs for serving the corresponding workload generated by the total number of hosts in each topology, that contribute on average one LLA request per second. Fig. 17 depicts the performance of the RevM, LFU, Self-tuning and Static strategies over the three ISP topologies using the non-uniform VM allocation. In the plots we include only the RevM policy performance in order to demonstrate its ability in trading idle time and QoS for achieving a highest revenue in the case of a sufficiently deployed number of VMs. We observe that RevM achieves better QoS, lower idle time, and significantly higher revenue in all three topologies than the rest of the strategies.

As expected, the QoS and idle time values remain consistent for each strategy across the three topologies, as a result of having similar proportion of computation budget to the total workload in each topology. On the other hand, the revenue obtained by each strategy is proportional to the amount of available VM resources. Consequently, each strategy attains the highest (lowest) revenue in AT&T (Exodus) topology.

Self-tuning performs close to RevM in terms of QoS gain, albeit obtaining a much lower revenue. As described before, in Self-tuning, Static and LFU strategies, cloudlets charge according to the lowest QoS gain of the accepted requests. Self-tuning strategy aims to select the highest QoS applications while keeping the utilisation of resources high. However, even Self-tuning strategy fails to set an appropriate threshold price that achieves a good balance between utilisation and revenue. On the other hand, RevM strategy is able to make a better selection of applications to accept and achieve both reasonably high utilisation and significantly higher revenue than the remaining strategies using both incremental and non-uniform resource allocation methods. Note that as decreasing the VM resources' budget the benefits of FogSpot are amplified.

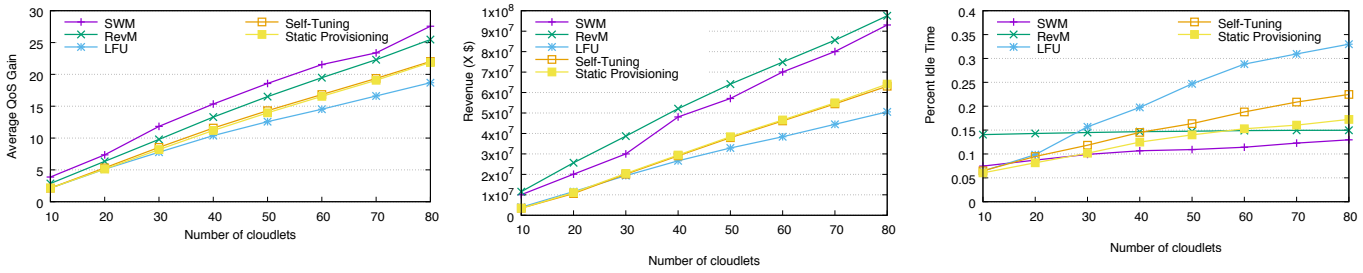
3) Impact of Engagement Duration: Next we investigate the impact of LLAs engagement duration on the FogSpot approaches performance. Specifically, we consider the case of varying *solely i*) the highest (H.) demanding LLA category, *i.e.,* $u_{\min} = 0$ and maximum QoS gain 100, and *ii*) the least (L.) demanding LLA, *i.e.,* $u_{\min} = 90$ and maximum QoS gain 10. That is, we vary the average engagement duration of only the most (or least) demanding LLA for each experiment while we keep the engagement time stable for the other 9 LLAs.

In Fig. 18 we consider a setting where the number of VMs is sufficient for serving the total number of requests in the small tree topology, *i.e.,* 70 VMs per cloudlet (as we see from the idle time captured in Fig. 15). Clearly, only changes in the duration of the most demanding LLA have an impact on the average QoS gain, *i.e.,* scenario H., for both SWM and RevM approaches (Fig. 18a). That said, the revenue is affected only for the RevM approach of scenario H., since scenario L. has no impact on the revenue as it is captured in Fig. 18b. On the other hand, the revenue of SWM remains stable since the supply of VMs exceeds the demand for resources.

The engagement duration impact is similar in a setting where the number of VMs per cloudlet is limited, *e.g.,*

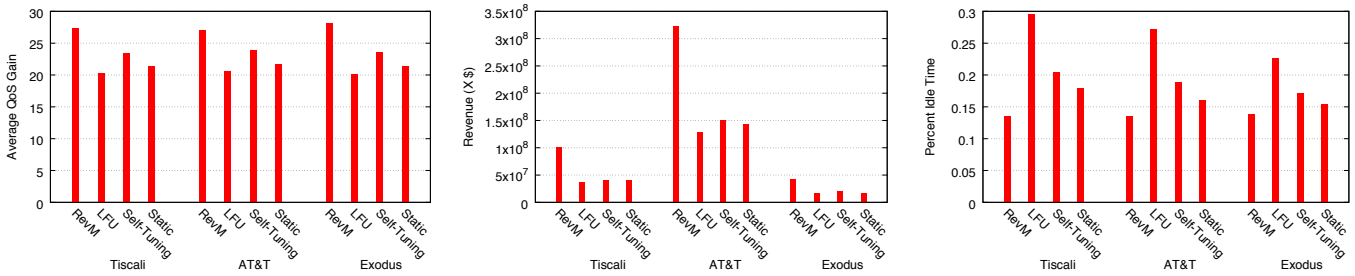
1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing



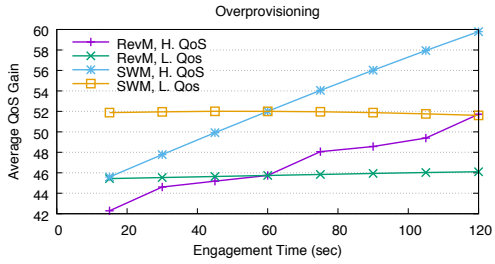
(a) QoS Comparison (b) Revenue Comparison (c) Idle Time Percentage Comparison

Fig. 16: ISP, incremental deployment of identical cloudlets of 40 VMs each.

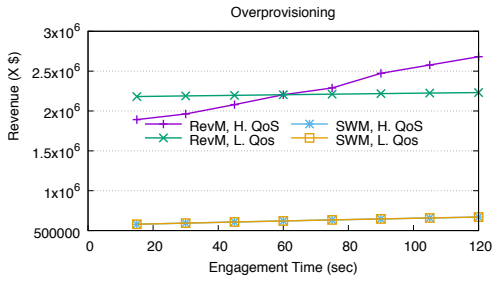


(a) QoS Comparison (b) Revenue Comparison (c) Idle Time Percentage Comparison

Fig. 17: 3 ISP topologies, Non-uniform Allocation of VMs.



(a) Avg. QoS Gain



(b) Revenue

Fig. 18: Small Tree Topology, Engagement duration Impact, Excess VMs availability, (Number of VMs 70).

10 VMs, since again the QoS gain is influenced only by changes in the most demanding LLA for both the SWM and RevM FogSpot variations. However, in that setting the most demanding LLA is also affecting the revenue of the SWM approach, Fig. 19, due to the fact that its requests are sufficient for utilising the cloudlet resources.[10]

[10]The ISP topology presented similar trends.

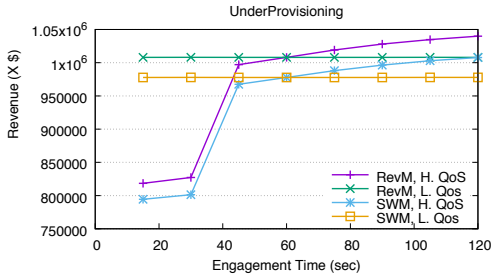


Fig. 19: Small Tree Topology, Engagement duration Impact on Revenue, Limited VMs availability, (Number of VMs 10).

VII. Conclusions

An increasing number of network applications with low latency requirements render cloud provisioning unfit for purpose. In this paper, we studied a market-based approach where applications lease third-party cloudlet resources for provisioning their instances. This is a promising research direction since it manages to provision applications closer to their users, for improving their perceived latency, while compensating cloudlets for their resource contributions.

Along these lines, we proposed FogSpot as an on-path, on-demand market based provisioning mechanism. FogSpot associates each cloudlet to a price that either targets to maximise cloudlet's resource utilisation or revenue. After that, requests forwarded in the network interact with cloudlets' prices in a way that if their gain, in terms of QoS improvement, exceeds that price they provision locally an application instance. FogSpot takes explicitly into account the provisioning of heavily stateful applications which one instantiated cannot be suspended and/or get migrated to another cloudlet location, without impairing irreversibly the perceived QoS. Our game theoretic analysis showed that FogSpot can derive the optimal

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2019.2895037, IEEE Transactions on Services Computing

prices in an uncoordinated way especially in hierarchical topologies. FogSpot advantages were demonstrated against a variety of proactive and reactive provisioning techniques in realistic topologies.

REFERENCES

[1] G. Aceto, A. Botta, W. De Donato, and A. Pescapè. Cloud monitoring: A survey. *Elsevier Computer Networks, 2013*.

[2] O. Agmon Ben-Yehuda et al. Deconstructing amazon EC2 spot instance pricing. *ACM Trans. on Economics and Computation, 2013*.

[3] O. Ascigil et al. On uncoordinated service placement in edge-clouds. In *IEEE CloudCom, 2017*.

[4] R. K. Balan et al. Simplifying cyber foraging for mobile devices. In *Mobile systems, applications and services*. ACM, 2007.

[5] M. V. Barbera et al. To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *IEEE INFOCOM, 2013*.

[6] S. Bhardwaj, L. Jain, and S. Jain. Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *ACM MCC, 2012*.

[8] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *IEEE HPPC, 2008*.

[9] M. Calder et al. Odin: Microsofts scalable fault-tolerant cdn measurement system. In *USENIX NSDI, 2018)*.

[10] B.-G. Chun et al. Clonecloud: elastic execution between mobile device and cloud. In *ACM Computer Systems, 2011*.

[11] M. Claypool and K. Claypool. Latency and player actions in online games. *Communications of the ACM*, 49(11):40–45, 2006.

[12] E. Cuervo et al. MAUI: making smartphones last longer with code offload. In *ACM Mobile systems, applications, and services, 2010*.

[13] F. R. Dogar et al. Decentralized task-aware scheduling for data center networks. In *ACM SIGCOMM CCR, 2014*.

[14] P. T. Endo et al. Resource allocation for distributed cloud: concepts and research challenges. *IEEE Network, 2011*.

[15] A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *ACM Theory of Computing, 2004*.

[16] G. P. Fettweis. The tactile internet: Applications and challenges. *IEEE Vehicular Technology Magazine*, 9(1):64–70, 2014.

[17] A. Greenberg et al. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM CCR, 2008*.

[18] S. Ha et al. Tube: time-dependent pricing for mobile data. *ACM SIGCOMM CCR, 2012*.

[19] L. Kleinrock. *Queueing systems, volume 2: Computer applications*, volume 66. wiley New York, 1976.

[20] R. Landa et al. Self-tuning service provisioning for decentralized cloud applications. *IEEE TNSM, 2016*.

[21] I. Lee and K. Lee. The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431–440, 2015.

[22] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic in-network caching for information-centric networks. In *ACM ICN, 2012*.

[23] R. W. Rosenthal. A class of games possessing pure-strategy nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973.

[24] L. Saino, I. Psaras, and G. Pavlou. Icarus: a caching simulator for information centric networking (icn). In *ICST SimuTools, 2014*.

[25] Z. Sanaei, S. Abolfazli, A. Gani, and M. Shiraz. SAMI: Service-based arbitrated multi-tier infrastructure for mobile cloud computing. In *IEEE ICCC, 2012*.

[26] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing, 2009*.

[27] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM CCR, 2002*.

[28] A. G. Tasiopoulos, O. Ascigil, I. Psaras, and G. Pavlou. Edge-MAP: Auction markets for edge resource provisioning. In *IEEE WoWMoM, 2018*.

[29] A. G. Tasiopoulos, O. Ascigil, I. Psaras, S. Toumpis, and G. Pavlou. On-path cloudlet pricing for low latency application provisioning. In *IEEE LANMAN, 2018*.

[30] L. Yang et al. A framework for partitioning and execution of data stream applications in mobile cloud computing. *ACM SIGMETRICS Performance Evaluation Review, 2013*.

[31] S. Yi, A. Andrzejak, and D. Kondo. Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *IEEE Transactions on Services Computing, 2012*.

[32] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. A. Lee, and J. Kubiatowicz. The cloud is not enough: Saving iot from the cloud. In *HotStorage*, 2015.

[33] L. Zhang, Z. Li, and C. Wu. Dynamic resource provisioning in cloud computing: A randomized auction approach. In *IEEE INFOCOM, 2014*.

[34] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Springer Journal of Internet services and Applications, 2010*.

[35] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang. How to bid the cloud. In *ACM SIGCOMM CCR, 2015*.

Argyrios G. Tasiopoulos is a PhD student in the Department of Electronic and Electrical Engineering, University College London, UK. He received the B.Sc. in Informatics and the M.Sc. in Computer Science, by the Department of Informatics of the Athens University of Economics and Business. His research efforts lies at the intersection of operations research, telecommunications, and economics. He is particularly interested in market designs for delivering the next generation of network applications.

Onur Ascigil received his PhD. degree from the Computer Science Department, University of Kentucky, Lexington, USA in 2014. From 2008 to 2014 he worked as a research assistant and from Jan. 2015 to Aug. 2015 as a Post-doctorate Research Associate at the Laboratory for Advanced Networking, University of Kentucky. In September 2015 he joined the Electronic and Electrical Engineering Department, UCL, London as a Research Associate.

Ioannis Psaras is an EPSRC Fellow at the Electrical and Electronic Engineering Department of UCL. He is interested in resource management techniques for current and future networking architectures with particular focus on routing, caching and congestion control. Before joining UCL in 2010, he held positions at the University of Surrey, and Democritus University of Thrace, Greece, where he also obtained his PhD in 2008. He has held research intern positions at DoCoMo Eurolabs and Ericsson Eurolabs.

Stavros Toumpis received the Diploma in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 1997, the M.S. degrees in Electrical Engineering and Mathematics from Stanford University, CA, in 1999 and 2003, respectively, and the Ph.D. degree in Electrical Engineering, also from Stanford, in 2003. He is currently an Assistant Professor at the Department of Informatics of the Athens University of Economics and Business, Athens, Greece. His research is on wireless ad hoc networks, with emphasis on capacity, network optimisation, and information theoretic issues.

George Pavlou is a Professor of Communication Networks in the Department of Electronic and Electrical Engineering, University College London, UK. He received a Diploma in Engineering from the National Technical University of Athens, Greece and M.S. and Ph.D. degrees in Computer Science from University College London, UK. His research interests focus on networking and network management. In 2011 he received the Daniel Stokesbury award for "distinguished technical contribution to the growth of the network management field".

1939-1374 (c) 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.