On-path Cloudlet Pricing for Low Latency Application Provisioning

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*—Cloud computing has been tremendously successful in providing a commercial infrastructure for hosting computationally intensive applications. Nevertheless, an increasing number of Low Latency Applications (LLAs) notably in the entertainment, IoT, and automative domains require response times much smaller than the supported ones by the typical "client-to-cloud" network model. Cloudlets have been introduced as "*data centres in a box*", for bringing computing resources "*closer*" to the end users. As a result, LLAs can take advantage of cloudlets to improve their Quality-of-Service (QoS) by reducing the underlying response times between their users and application instances' location. In this work, we study the emerging market of stateful LLAs' provisioning over geo-distributed third-party cloudlets. We assume that cloudlets offer their resources in the form of Virtual Machines (VMs) via collocated markets. Forwarding requests for LLAs interact with cloudlet markets for performing on-path and on-demand resource provisioning. We introduce a pricing scheme where users pay a fixed price for each time unit of their engagement to an LLA instance. Our evaluation on realistic topologies and application requests demonstrate the merits of on-demand provisioning when accompanied by a pay-as-you-go pricing scheme.**

I. Introduction

Cloud computing is the prominent cost-efficient infrastructure for delivering computationally demanding applications to end-users. Clouds' abundance of resources enables them to elastically cope with changes in applications' demand in a scalable manner via on-demand computation. That is, Clouds exploit economies of scale for decreasing their running cost [7] while deploying usage-based pricing mechanisms for controlling the demand for their computing resources. However, an increasing number of applications (like augmented reality, automotive, health monitoring etc.) require low response times, rendering the current cloud-based infrastructure unfit for the purpose of application provisioning. 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).

Cloudlets have been proposed as "data centres in a box" that can be deployed throughout the Internet, for bringing computing resources *closer* to the end users [12]. That is, cloudlets have the potential of improving the network conditions by decreasing the Round-Trip-Time (RTT) between end-users and application instances when hosted at cloudlets' points-of-presence. Clearly, cloudlets can serve as an alternative computing infrastructure, that augments as well as complements the centralised provisioning at the cloud, for enabling LLAs.

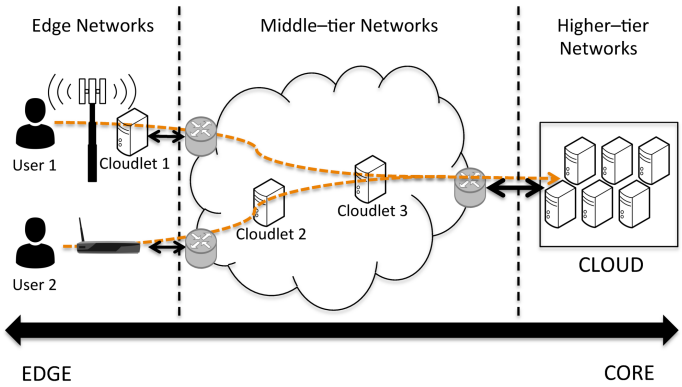


Fig. 1: On-path application provisioning setting

Nevertheless, cloudlets are incapable of providing the essentially boundless elasticity of the cloud, meaning that occasionally the demand for resources at a given cloudlet can exceed their availability. That is, resource allocation mechanisms have to prioritise the requests that benefit the most when served by a cloudlet. Especially in the case of stateful LLA provisioning, resource allocation is a challenging task since once a request is accepted and a user is engaged to an application instance, the established application session cannot be disrupted for reallocating the involved resources. The reason is that session disruptions can cause a user to quit the LLA, which is considered the worst possible outcome in terms of QoS.

In this work we investigate the emerging market of provisioning heavily stateful LLAs over third party cloudlets. In our context, heavily stateful LLAs are defined as the ones that cannot be migrated to another cloudlet location, due to the size of their runtime data generated by their users, without seriously damaging their QoS [4]. We consider that computation is available either at the edge or middle-tier locations of the network, in the form of cloudlets, and/or at distant clouds/data centres (Fig. 1). We argue that application 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 cloudlet resources be allocated over time to different applications/services?* and *ii*) *how much should a cloudlet charge an application?* Here, we present a decentralised pricing mechanism that answers

both questions, while addressing the challenges of dynamic application provisioning over an infrastructure of third-party cloudlets.

We introduce a pricing mechanism that associates each cloudlet with a price for on-path, on-demand, uncoordinated LLA provisioning. In our context, cloudlets offer their resources in the form of Virtual Machines (VMs) via collocated markets. As LLA requests are forwarded towards a default execution location, *i.e.,* at the Cloud, they interact with on-path cloudlet markets. If the price of a market is below the gain the LLA would have if it is served, 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 requests of User 1 (User 2) attempt to provision resources on-path at Cloudlet 1 (Cloudlet 2) followed by Cloudlet 3 before getting served by the Cloud as the last available provisioning infrastructure. Our design addresses explicitly the heavily stateful requirement of seamless users' engagement to LLA instances while setting the price of each cloudlet based on its resource utilisation. To this end, the main technical contributions of this paper are as follows:

1) We study the emerging market of heavily stateful LLAs' provisioning over independent cloudlets from an economics point of view.
2) We develop a pricing mechanism for on-path, on-demand LLA provisioning, taking into account the cloudlets' computing resource limitations and the end users' requirement for seamless LLA engagement.
3) We evaluate the merits of our mechanism in an ISP topology under trace-based application requests.

II. Design Rationale & System Model

A. Design Rationale

We aim to design a market mechanism tailored to the provisioning of LLAs over an uncoordinated cloudlet infrastructure. Our design has to address explicitly the application provisioning challenges of i) cloudlets' resource discovery, since cloudlets points-of-presence are expected to exceed by far the number of clouds, and ii) the cloudlets' limited elasticity of resources.

Given that application requests are forwarded in the network towards a distant data centre, we argue for both **on-path** and **on-demand** application provisioning. That is, by applying on-path provisioning, there is no need for a centralised resource discovery process since resources are discovered in real-time opportunistically. Furthermore, due to on-demand provisioning, cloudlets do not waste resources since each allocated VM serves an application request. We argue that these design choices enable the most promising and incrementally deployable conditions for the problem of application provisioning we tackle; providing an uncoordinated solution. However, on-demand provisioning has to be accompanied by a pricing scheme that controls the demand at each cloudlet, and this needs to happen in a way that the requests that have the highest provisioning gain at a specific location are given priority.

System Model	
$\mathcal{S}$	Set of LLAs.
$\mathcal{D}$,D_s	Set of cloudlets, default data centre of LLA s.
$\mathcal{P}$	Set of requests' access points.
$u^d_{s,p}$	Per second QoS gain of LLA s at cloudlet d for request from access point p, in terms of network condition.
Charging Mechanism	
π_d	Price of cloudlet d.
$b^d_{s,p}$	Bid of request produced at access point p for LLA s at cloudlet d.
$\rho_d^{\min}, \rho_d^{\max}$	Target minimum and maximum utilisation of resources at d.
$\Delta\pi$	Price decrease step.

TABLE I: Pay-as-you-go Notation

Cloud resources are offered in terms of remote instances, *i.e.,* virtual machines (VMs), with dedicated CPU, memory, and storage resources, as an Infrastructure-as-a-Service (IaaS) [2]. For example as a cloud resource provider, Amazon offers instances under three pricing schemes:

1) *Reserved instances* guarantee the long-term availability of an instance, *i.e.,* for more than a year, by charging fixed usage-based prices.
2) *On-demand instances* guarantee the short-term availability of an instance, *i.e.,* for an hour, by also charging fixed usage-based prices.
3) *Spot instances* create an auction-based market for *spare* instances. In detail, users can use spot instances only if their bids exceed a spot price while spot prices are updated every 5 minutes; meaning that the availability of instances is not guaranteed.

Offering cloudlet resources as reserved and/or on-demand instances fails to capture demand fluctuations under the bounded instances elasticity. Furthermore, 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. Despite some efforts in strategically bidding spot instances for uninterrupted tasks to the cloud [16], we suggest that *a cloudlet pricing plan should include the applications' seamless execution in its inherent characteristics*, especially in the case of heavily stateful applications. In this work we argue that cloudlets' pricing schemes should follow a pay-as-you-go structure in terms of application user engagement duration. In other words, the LLA providers should be charged based on the time their users occupy a cloudlet instance.

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. We assume that each LLA $s \in \mathcal{S}$ can serve its users demand at a distant cloud/data centre, denoted by D_s, whose capacity is sufficient for serving the total number of LLA requests it receives. Application users are connected to the network via a set of $\mathcal{P} \triangleq \{1, 2, ..., P\}$ access points. We assume that the network conditions to a cloudlet are access-point specific. That is, a user connected to access point p while requesting an LLA $s \in \mathcal{S}$ 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 $d \in \mathcal{D}$, as opposed to the default execution cloud D_s. The notation followed in throughout the paper is presented in Table I.

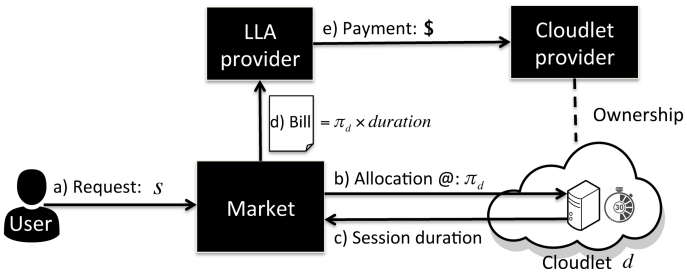


Fig. 2: On-Demand Provisioning Pay-as-you-go Charging Scheme Overview

III. Pay-as-you-go Mechanism

In this section, we describe the individual components of the proposed pay-as-you-go mechanism, namely *i*) the on-demand provisioning component, and *ii*) the utilisation-based price derivation component. Given a price assignment at each cloudlet, the on-demand provisioning component defines the real-time interaction of LLA requests with the collocated cloudlets' markets, as the requests are forwarded in the network. On the other hand, the price derivation component is responsible for deriving a price for each cloudlet given the resource utilisation achieved at each cloudlet.

A. On-Demand Provisioning

An overview of the 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 the request with each market is characterised by the following properties:

A1) The request is for a single VM.
A2) The request is associated with a bid, $b^d_{s,p}$, that expresses the user's willingness to pay for a VM for each engagement time-unit of the user to an LLA s instance at d when her access point is p.
A3) The request is not queued at the market, *i.e.,* the request is either served or rejected immediately.
A4) The VM allocation time overhead has no impact on the LLA's QoS.

In more detail, let the price for the use of a VM at cloudlet d be π_d. Upon the arrival of the request the market operates according to the following rules:

R1 If there are no available VMs, the request is rejected.
R2 Else, if there are available VMs:

- If $b^d_{s,p} \geq \pi_d$, a VM is allocated to serve the request at price π_d for each time unit of user engagement.
- Else, if $b^d_{s,p} < \pi_d$, the request is rejected.

Assuming $b^d_{s,p} \geq \pi_d$ and the existence of available VMs, the allocation is taking 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.[1] Next, we explain how the request bids as well as the cloudlet prices are derived.

B. Request Bids and Cloudlet Prices derivation

Let $u^d_{s,p}$ be the per-time-unit QoS gain of LLA s when a request is coming from access point p to get served at cloudlet d, instead of its default cloud. Bidding truthfulness is defined in the following straightforward way:

Definition 1. *A bid $b^d_{s,p}$ for a request arriving at cloudlet d from access point p for LLA s 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 the cloudlet's price, π_d, the net utility rate of the involved request is defined as:

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

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

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

Proof. We investigate the following possible cases:

- If $u^d_{s,p} \geq \pi_d$ then the bidder would win the item with a truthful bid as well as an overbid, *i.e.,* $b^d_{s,p} \geq u^d_{s,p}$.
- If $u^d_{s,p} \geq \pi_d > b^d_{s,p}$ underbidding returns a net utility rate equal to 0, as opposed to a truthful bid for which the net utility rate is non-negative.
- If $\pi_d > u^d_{s,p}$ then the bidder would loose the item with a truthful bid as well as an underbid, *i.e.,* $u^d_{s,p} \geq b^d_{s,p}$.
- If $b^d_{s,p} > \pi_d > u^d_{s,p}$, then overbidding would return a negative net utility rate, as opposed to a truthful bid for which the net utility rate is 0.

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

Because the bids must be truthful for a rational user, the strategy of their deployment is straightforward, *i.e.,* assigning a price $b^d_{s,p} = u^d_{s,p}$ to each request. Therefore, a request has to simply be associated to its involved QoS per time unit gain at a given cloudlet.

Lastly with respect to cloudlet price derivation, we assume that each cloudlet d periodically (every w seconds) observes the utilisation ρ^{old}_d of its resources that was achieved at the previously set price π^{old}_d. Then, given a minimum target utilization $\rho^{\min}_d$ the price is adjusted according to:

$$\pi_d = \begin{cases} \pi^{\text{old}}_d - \Delta\pi, & \text{if } \rho^{\text{old}}_d < \rho^{\min}_d, \\ \pi^{\text{old}}_d + \Delta\pi, & \text{if } \rho^{\text{old}}_d > \rho^{\max}_d, \\ \pi^{\text{old}}_d, & \text{otherwise.} \end{cases}$$

[1]The technical details of the explained process, related to the security, verification, etc., are beyond the scope of this paper although orthogonal to its contribution.

where $\Delta\pi > 0$ is the price decreasing increment. Note that as a price decreases (increases) the number of requests that can be served at cloudlet d rises (drops) with the result of increase (decrease) in the utilisation of d's resources.

IV. Performance Evaluation

In this section, we demonstrate Pay-as-you-go's performance via simulations in an ISP topology. We implement the Pay-as-you-go and the baseline approaches, described next, by extending the Icarus caching simulator for application provisioning problems [11].

Application categories for LLAs: The QoS of the LLAs is expressed as a decreasing function of the end users' perceived latency to each cloudlet [5], [15]. Similarly to [8] for abstract resource allocation gains, we consider that each LLA is characterised by a decreasing QoS function of the latency x having the general form:

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

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 $u_{\max} = 100$ and $l_{\min} = 5$ ms[2] when is not specified differently. Moreover, the 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 $l_{\min}$.

Based on Eq. 1, we create ten *LLA categories*, each associated to a QoS function that models a certain sensitivity of the LLA to latency. For each LLA category we assign a different $u_{\min}$ value from the set of $\{0, 10, 20, \ldots, 90\}$. In this way, distinct application categories have diverse QoS gains from being provisioned at a cloudlet, varying, for example, from 10, for $u_{\min} = 90$ and less-latency sensitive LLAs, to 100, for $u_{\min} = 0$ and latency-critical LLAs, when provisioned at the edge, *i.e.,* $x = l_{\min}$. We consider ten LLA categories to be sufficient for the purpose of our evaluation since this number is comparable to the currently considered types in the context of IoT [3], [10] and Tactile Internet [6]. Lastly, we assume that users remain engaged to their LLA instance on average for 1 minute while we execute the following experiments for the duration of three hours in our simulator.

LLA Realistic Requests' Generation: Each user selects one of the 10 LLA categories based on the sequence of request arrivals in Google's cluster dataset[3]. In detail, we associate each LLA category to a "ParentID" field, that identifies the service, of the 10 most popular services in the dataset accounting for more than 200K requests. Then by selecting random time intervals in the period of the seven hours that the dataset covers, users request LLA categories based on the sequence of "ParentID" fields that arrive into Google's cluster.

[2]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

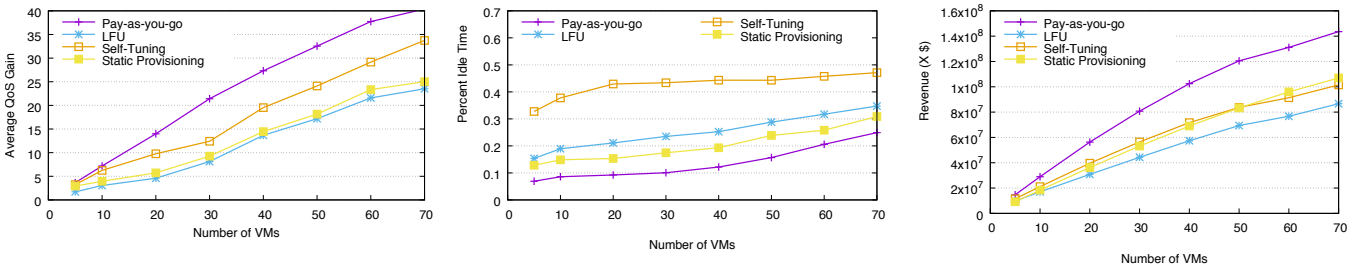
[3]Available at https://research.googleblog.com/2010/01/google-cluster-data.html.

ISP Topology & Cloudlets Deployment: We evaluate Pay-as-you-go under the Tiscali topology of the Rocketfuel dataset [13] that contains 240 nodes and 404 links. Among the 240 nodes, we designate as hosts the 80 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 belong to at least a single shortest path from a host to the Cloud. That is, we have 79 cloudlets all over the topology for reducing the Cloud RTT latency that on average is 155 ms to each host. The default number of available VMs per cloudlet is 20, while we set the minimum and maximum target utilisation of each cloudlet to be equal to $\rho_d^{\min} = 0.85$ and $\rho_d^{\max} = 0.95$ respectively.

Comparison approaches: We compare Pay-as-you-go against the following approaches:

- ***Static Provisioning*:** Cloudlets assign their VMs to application instances according to their demand that is considered known in advance, *e.g.,* an equal number of VMs is assigned to each LLA in the case of an uniform requests' distribution.
- ***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 application, and so on, until all the VMs at a cloudlet are allocated. The number of VMs assigned to each application is determined according to the number of both the requests served and rejected.
- ***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 a number of unallocated VMs to the current application. This results with VMs being assigned to applications that are willing to *pay the most*. The number of VMs assigned to each application again depends on its observed demand.

The Static approach is a form of *proactive* provisioning, because it relies on prior knowledge of the requests demand 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 default length of the observation period to 5 minutes, *i.e.,* 300 seconds. Note that we consider a single LLA per LLA category in order to give an advantage to the above-mentioned approaches against Pay-as-you-go'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



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

Fig. 3: The impact of the number of VMs on the performance of the provisioning mechanisms.

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 Pay-as-you-go, we assume a charging mechanism, identical to Pay-as-you-go, 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 the requests' bidding truthfulness since they would pay different prices for identical VMs.

A. Impact of the Number of VMs

In Fig. 3, we depict the performance of the LLA provisioning approaches for an increasing number of VMs per cloudlet for the metrics of: i) users' average QoS gain ($u(x)$), ii) percentage of idle time experienced by the VMs of the cloudlets, and iii) the total revenue obtained by all cloudlets. User requests served by the back-end Cloud experience the maximum latency (i.e., $l_{\max}$), and thus obtain zero gain. The revenue obtained by a cloudlet is computed as the product of the engagement time of the served users at each VM and the per-second usage price of the VM determined by a cloudlet. For the Pay-as-you-go mechanism, the per-second usage price of the VMs at a cloudlet is derived periodically based on the difference between the observed and the target utilisation, as described in Section III-A. For the rest of the strategies, we assume a second-price auction mechanism, where the winning bidders pay for the next highest bid, and the bids are set to the QoS gain of the users as in a true value bidding.

As shown in Fig. 3a, the Pay-as-you-go mechanism can achieve a higher average QoS gain than the other strategies. Specifically, the Self-Tuning mechanism is outperformed since it ignores the utilisation of resources when provisioning the LLAs. As a result, a larger portion of the requests are served by the Cloud without obtaining any QoS gain. On the other hand, LFU and Static approaches provision LLAs purely based on utilisation, without considering the QoS gain of the users, resulting in lower average QoS gain.

Our QoS findings are in accordance with the average idle time percentage of the resources, as it is demonstrated in Fig. 3b. As expected, Static provisioning presents a higher utilisation of resources, *i.e.,* lower idle time, compared to LFU and Self-tuning by exploiting the knowledge of LLAs' upcoming demand. However, since Static, LFU, and Self-tuning approaches operate by assigning VMs to LLA instances periodically, they are outperformed by the Pay-as-you-go mechanism that provisions requests as soon as they arrive, in an on-demand manner.

As a result of achieving higher QoS gain and utilisation of resources, the Pay-as-you-go mechanism obtains the highest revenue, as depicted in Fig. 3c. As the number of VMs increases, the idle time of the strategies raises by around 15%. At the same time the average QoS gain of the users presents an upward trend, which leads to higher bids for resources followed by greater revenues. We observe that the Self-Tuning approach obtains higher revenue than the LFU and nearly the same revenue as Static provisioning.

B. Impact of Observation Period Length

In the experiments presented earlier, we used a default observation period length of 300 seconds (5 minutes). In Fig. 4, we demonstrate the impact of this parameter on QoS gain (left plot) and idle time percentage (right plot) of the provisioning mechanisms.

First of all, we observe that the performances of both LFU and Self-tuning approaches are increasing function of the observation period duration before being stabilised at 200 seconds. This is related to the fact that LFU and Self-tuning rely on the estimation of LLAs' demand, that is inaccurate for short observation durations in the trace of requests we are using since it fails to capture the demand fluctuations.

In terms of QoS gain, the Self-tuning mechanism achieves a remarkable increase of 33% in the duration interval 0 to 200 seconds. At the same time, the LFU mechanism achieves nearly 50% increase in resource utilisation since the idle time percentage drops from 40%, for a duration of 0 seconds, to 20%, for 200 seconds. The Pay-as-you-go mechanism is affected the least by the observation period factor while constantly achieving the highest QoS gain and lowest idle time percentage compared to the rest of approaches.

Our evaluation demonstrates that the Pay-as-you-go mechanism can provision resources with high utilisation while providing high QoS gains for the end-users. The price derivation mechanism of our on-demand provisioning mechanism leads to higher revenues than the provisioning mechanisms that periodically allocate the VMs to LLAs based on the observed demand.

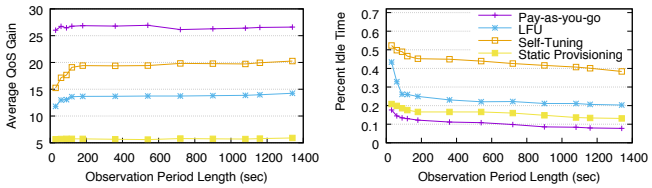


Fig. 4: Impact of observation period length on QoS gain of users (left) and average idle times of VMs (right).

V. Related Work

Cloudlets have been proposed in [12] with the initial purpose of acting as a surrogate infrastructure where mobile devices can offload intensive tasks as a response to their computing and battery limitations. Specifically, [1] addresses the problem of application-specific task offloading over the fog infrastructure, *i.e.,* a task requires the corresponding virtual instance of the application at a cloudlet before getting offloaded. However, the authors only consider stateless applications, while ignoring completely the economic aspects of such a problem. In [9], the authors propose a self-tuning application provisioning mechanism using a 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 and takes place in a centralised way that challenges the scalability of such a scheme. Closer to our setting, [14] discusses LLA provisioning over third-party cloudlets in an economic context for "lightly" stateful applications, which can be migrated between cloudlets without severely degrading the involved QoS for mobile users (as a result of the hand-offs between different cells). Similar to spot pricing mechanisms (*e.g.,* [16], our approach can react to transient changes in demand, but without either suspending applications or collecting (*i.e.,* queuing) user bids to execute an auction.

VI. Conclusions

An increasing number of network applications with low latency requirements render cloud provisioning unfit for purpose. In this paper, we investigated the emerging market where heavily stateful low latency applications lease third-party cloudlet resources for provisioning their instances. This is a promising research direction for bringing application instances closer to their users in order to improve the involved perceived latency.

Along these lines, we introduced a Pay-as-you-go charging scheme as an on-path, on-demand market based provisioning mechanism. In the proposed scheme, cloudlets are associated with a single price that is adjusted based on their target resource utilisation. Then LLAs' requests forwarded in the network interact on-path with collocated markets to the cloudlets, provisioning on-demand an application instance to the first cloudlet with a price lower than their QoS gain. Our mechanism address explicitly the challenges of heavily stateful applications whose instances cannot be suspended and/or migrated to other cloudlet locations, without impairing irreversibly the perceived QoS. The advantages of such an approach were demonstrated against a variety of proactive and reactive provisioning techniques in realistic topologies.

Acknowledgment

This work has been supported by the EC H2020 ICN2020 project (GA no. 723014), the EPSRC INSP fellowship (EP/M003787/1), and UK EPSRC KCN project (EP/L026120/1).

References

[1] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou. On uncoordinated service placement in Edge-Clouds. In *Cloud Computing Technology and Science (CloudCom), 2017 IEEE International Conference on*, pages 41–48. IEEE, 2017.

[2] 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.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the Internet of Things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

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

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

[7] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[8] H. Izakian, A. Abraham, and B. T. Ladani. An auction method for resource allocation in computational grids. *Future Generation Computer Systems*, 26(2):228–235, 2010.

[9] R. Landa, M. Charalambides, R. G. Clegg, D. Griffin, and M. Rio. Self-tuning service provisioning for decentralized Cloud applications. *IEEE Transactions on Network and Service Management*, 13(2):197–211, 2016.

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

[11] L. Saino, I. Psaras, and G. Pavlou. Icarus: a caching simulator for information centric networking (icn). In *Proceedings of the 7th International ICST conference on Simulation Tools and Techniques*, pages 66–75. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[12] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for VM-based Cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[13] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.

[14] A. G. Tasiopoulos, O. Ascigil, I. Psaras, and G. Pavlou. Edge-MAP: Auction markets for edge resource provisioning. In *IEEE 19th World of Wireless Mobile and Multimedia Networks (WoWMoM)*, 2018.

[15] 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.

[16] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang. How to bid the cloud. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 71–84. ACM, 2015.