A Decentralized Economic Load-Balancing Framework among Federated Cloudlets

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

18-08-2021 / 27-08-2021

CITATION

Mondal, Sourav; Das, Goutam (2021): A Decentralized Economic Load-Balancing Framework among Federated Cloudlets. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.15222867.v1>

DOI

[10.36227/techrxiv.15222867.v1](https://doi.org/10.36227/techrxiv.15222867.v1)

A Decentralized Economic Load-Balancing Framework among Federated Cloudlets

Sourav Mondal, *Member, IEEE*, Goutam Das, *Member, IEEE*, and Elaine Wong, *Senior Member, IEEE*

Abstract—Edge computing servers like cloudlets from different service providers compensate scarce computational and storage resources of mobile devices, are distributed across access networks. However, the dynamically varying computational requirements of associated mobile devices make cloudlets either overloaded or under-loaded. Hence, load balancing among neighboring cloudlets appears to be an essential research problem. Especially, the load balancing problem among federated cloudlets from the same as well as different service providers for low-latency applications needs significant attention. Thus, in this paper, we propose a decentralized load balancing framework among federated cloudlets for low-latency applications that focuses on latency bound rather than latency minimization. In this framework, we employ dynamic processor slicing for handling heterogeneous classes of job requests. We propose a continuous-action reinforcement learning automata-based algorithm that enables cloudlets to independently compute the load balancing strategies in a completely distributed network setting without any exhaustive control message exchange. To capture the economic interaction among federated cloudlets, we model this load balancing problem as an economic and non-cooperative game and by scaffolding the properties of the game formulation, we achieve faster convergence of the reinforcement learning automata. Furthermore, through extensive simulations, we study the impacts of exploration and exploitation on learning accuracy.

Index Terms—Cloudlets, load balancing, long short-term memory, non-cooperative game theory, reinforcement learning.

I. INTRODUCTION

THE next-generation Internet is not only expected to route data, but also to store and process data generated from a large number of pervasive mobile devices like smartphones and Internet-of-Thing (IoT) devices. With the recent emergence of ultra-reliable and low-latency communication (uRLLC) applications such as virtual/augmented reality, automotive, and tele-operation as part of the tactile internet [1], [2], edge computing solutions like multi-access edge computing, fog computing, and cloudlet computing [3] have been proposed to reduce both communication and computation latencies. Edge computing servers like cloudlets are fundamentally a computer or cluster of computers installed in the proximity of mobile device users and distributed across access networks [4]. As cloudlet computing systems are essentially distributed computing systems, therefore, various efforts have been made to design a suitable job request allocation problem

from mobile devices to cloudlets while meeting computation and communication constraints [5]–[11]. Although job allocation frameworks allocate job requests to the most favourable cloudlets, due to the dynamic nature of job request arrival process, cloudlets at different parts of a large network become overloaded and/or under-loaded at different times. Thus, a plethora of research initiatives are present in the literature to achieve an efficient load balancing frameworks among neighbouring cloudlets [12]–[17].

Primarily, centralized and decentralized control mechanisms are used in existing literature to address load balancing problems [18]. A common objective function and a series of constraints are formulated in *centralized optimization models* to determine the optimal load balancing strategies for all cloudlets. A centralized problem of latency minimization is formulated by the authors of [12] and proposed a network-flow based heuristic algorithm for solving it. On the contrary, in *decentralized control models*, all distributed nodes exchange their local control information among themselves and determine the load balancing strategies without any central controller node. The authors of [19] made feasibility study of two decentralized load balancing algorithms. Although such models are robust for large networks, they cause an exhaustive control message exchange and communication burden on the network. To avoid this, reinforcement learning algorithms appear to be a valuable approach but can present different complexity and convergence issues in real-time [20]. In [21], the authors proposed an efficient reinforcement learning algorithm to find the optimal load balancing decision for fog nodes with unknown reward and transition functions. The authors of [22] proposed a deep recurrent Q-network approach to approximate the optimal joint task offloading and resource allocation for heterogeneous job requests in multi-fog node systems.

Recently, there is a growing interest in applying cooperative and non-cooperative game-theoretical models to various network-related issues, as game theory offers many effective tools for evaluating and researching the relationship between distributed agents in conflict and cooperation [23]. The authors of [14] proposed a distributed non-cooperative load balancing game in small cell networks among the neighboring cloudlets, and compared its findings with a centralized load balancing system that leverage the Lyapunov-drift technique to maximize the long-term system performance. Each cloudlet tries to minimize end-to-end latency costs under specific energy and latency constraints in this formulation. This model, therefore, works very well if the network is loaded moderately, but under very high load conditions it performs very poorly because under very high load conditions, some of the cloudlets start

S. Mondal and E. Wong are with the Department of Electrical and Electronic Engineering, The University of Melbourne, VIC 3010, Australia. (e-mail: smondal@student.unimelb.edu.au, ewon@unimelb.edu.au).

G. Das is with the G. S. Sanyal School of Telecommunications, Indian Institute of Technology Kharagpur, 721302, India. (e-mail: gdas@gssst.iitkgp.ac.in).

to violate the latency constraints and the Nash equilibrium (NE) solution becomes infeasible. By identifying the estimated latency as the dis-utility function of every cloudlet, the authors of [16] devised a non-cooperative load balancing game where cloudlets try to minimize its disutility and proposed an iterative proximal algorithm to compute the NE solution. In this framework, none of the cloudlets is allowed to offload until their incoming job requests reach a certain threshold. Nonetheless, this algorithm tends to assign a large number of job requests to the under-loaded cloudlets and hence, the end-to-end latency overshoots under very high load conditions.

In general, we notice that most of these load balancing frameworks assume that all the cloudlets are from a single service provider (SP). However, in a practical deployment scenario, usually multiple cloud SPs install cloudlets over the same customer base and to the best of our knowledge, there is no existing cloudlet federation frameworks for controlling and federating cloudlet resources across multiple SPs [3]. Therefore, in this paper we focus on load balancing problem among *federated cloudlets*, defined as a set of neighboring cloudlets from the same as well as different SPs. Network optimization based frameworks proposed in [12]–[14], [16] are inapplicable against such scenarios because cloudlets from different SPs are *self-interested agents* that are most-likely to deviate from a global optimal solution. Moreover, a cloudlet needs to *pay incentives* for offloading excess job requests to a neighboring cloudlet of different SP, but no payment is required for offloading to a cloudlet of the same SP. Thus, to capture this multi-party interaction among federated cloudlets, we formulate the load balancing problem as an economic and non-cooperative game that acts as an optimization problem among cloudlets from the same SP and acts as a non-cooperative game among cloudlets from different SPs.

We critically observe that most of the existing works in literature stressed on minimizing the overall latency while addressing load balancing problems among neighboring cloudlets. However, we feel that keeping the overall latency bounded within the requested quality-of-service (QoS) latency target is more important for economic interaction among federated cloudlets. For example, with a QoS latency target of 10 ms, users do not differentiate among job processing times 4 ms, 8 ms, or 10 ms. Thus, under-loaded cloudlets will always encourage its neighboring overloaded cloudlets to offload extra job requests to them because this provides an opportunity to earn extra incentives. Nonetheless, failing to meet the QoS latency target should incur a significant economic penalty on the cloudlets. With this realization, we propose a novel economic utility function for the cloudlets which is maximum when the end-to-end latency is equal to the QoS latency target for each job class and decreases very rapidly if the overall latency exceeds the QoS target latency.

As the existing load balancing frameworks make the load balancing decisions after the actual job request arrival to cloudlets, the overhead time of the load balancing algorithms makes these frameworks highly unfit, especially for low-latency applications. To deal with such scenarios, we make the cloudlets predict the job request arrival rates and make the load balancing decisions beforehand, so that immediate processing

after actual job request arrival is possible. Furthermore, most of the works assume that job requests demand similar computational resources and do not address the processing of multi-class job requests. However, in practice job requests from different users are heterogeneous and may demand a diverse set of computational resources. Therefore, in this work we consider a general scenario where incoming job requests are from heterogeneous classes and we consider that the cloudlets use *processor slicing* technique to dynamically slice its computational resources over time for handling heterogeneous job classes. Both these aspects are aligned with beyond 5G and 6G network visions [24].

To compute the pure strategy NE load balancing strategies of the federated cloudlets, we propose a decentralized framework in this paper. Although distributed frameworks are more robust than centralized frameworks, all the cloudlets need to exchange extensive control information among themselves [18], [25]. This issue can be resolved by using various artificial intelligence-based schemes to learn network conditions and make load balancing decisions. However, sometimes the job request arrival process may become fast-varying with low-correlation between trained data and real-time data. Hence, we avoid static supervised learning methods to find the NE load balancing strategies that heavily rely on historical data. We propose a *reinforcement learning automata-based algorithm* while ensuring quick convergence by using the properties of the underlying game. This empowers the cloudlets to make load balancing decisions independently, with a minimal interchange of control messages.

Note that cloudlets are clusters of computers and they are located at some distance from each other. Therefore, along with the processing latency, the intermediate data transmission latency also plays a crucial role in load balancing problems. Most load balancing frameworks do not pay significant attention to this factor. Nonetheless, depending on the bandwidth availability in the inter-cloudlet links, the computation offloading strategies of the cloudlets may also get affected. Hence, We address this issue by considering a bandwidth constraint in our game formulation. Our primary contributions in this paper are given as follows:

- (i) We formulate the load balancing problem among federated cloudlets with heterogeneous classes of job requests as a novel economic and non-cooperative game-theoretic problem and prove the existence of NE of this game formulation.
- (ii) We consider practical bandwidth constraints for inter-cloudlet links in our game formulation and design a complete time-slotted model for implementation of the proposed load balancing framework in practice.
- (iii) We propose a distributed continuous-action reinforcement learning automata-based algorithm such that neighboring cloudlets can independently compute the NE load balancing strategy and quick convergence is ensured by scaffolding the learning algorithm with the particular characteristics of the underlying load balancing game.
- (iv) We study various characteristics of the NE learning algorithm against a realistic setting by using the cluster-usage traces released by Google in 2011. This trace has

non-stationary and self-similar characteristics, but we can still achieve a very high degree of accuracy in learning the NE strategies.

- (v) Finally, we show that any participating cloudlet can achieve better economic utilities by following our proposed NE load balancing strategies than recently proposed game-theoretic load balancing frameworks, particularly under highly overloaded conditions.

The rest of this paper is organized as follows. In Section II, the details of the system model are presented. In Section III, a non-cooperative game-theoretic problem among federated cloudlets for computation offloading is formulated. In Section IV, a distributed continuous-action reinforcement learning automata-based algorithm is proposed. In Section V, the proposed load balancing framework is evaluated. Finally, in Section VI, our primary achievements by using the game-theoretic framework are summarized.

II. SYSTEM MODEL

In this section, we discuss the considered system model and the primary assumptions made. We consider a general heterogeneous deployment scenario for federated cloudlets over access networks. The total number of federated cloudlets in the network is $N \geq 2$ and $C = \{1, 2, \dots, N\}$ denotes the set of all federated cloudlets. These cloudlets receive heterogeneous job requests from their respective connected devices. If any cloudlet becomes overloaded, it can offload the extra load to its neighboring under-loaded cloudlets, as shown in Fig. 1.

(a) Job request arrival process: We consider multiple classes of job requests with heterogeneous QoS requirements and denote the average job request arrival rate of class- $m \in \mathcal{J} = \{1, 2, \dots, M\}$ to the i^{th} cloudlet by λ_i^m . The total job request arrival rate at the i^{th} cloudlet is given by $\lambda_i = \sum_{m=1}^M \lambda_i^m$. Each cloudlet relies on the access network SPs to successfully deliver the job requests from the associated mobile devices to them and pays for the bandwidth consumed. However, if the network fails to deliver some of the job requests to the cloudlets due to bandwidth constraints, then the network SPs pay a penalty in proportion to the undelivered job requests. As the average job request arrival varies from time instance to time instance, we assume that each λ_i^m is *independently distributed* over the support $\Lambda_i^m = [0, \lambda_i^{m, \max}]$, $\forall i \in C$. Therefore, the *computation job request profile* or *true type* of all the federated cloudlets is represented as $\lambda^m = (\lambda_1^m, \lambda_2^m, \dots, \lambda_N^m) \in \Lambda^m = (\Lambda_1^m \times \Lambda_2^m \times \dots \times \Lambda_N^m)$. In practice, the job request arrival process to cloudlets is *self-similar* and *non-stationary* [26]. Therefore, the cloudlets predict the incoming job request arrival rates by employing long short-term memory (LSTM) networks [27]. The transmission latency of the incoming job requests and the intermediate transmission latencies with the neighboring cloudlets are also estimated by each cloudlet.

(b) Job request service process: We assume that processors in the cloudlets have similar job processing capabilities, i.e., the same CPU cycle rate (cycles/s), say F_i^m for each job class- m . We also consider that the job requests from each class- m require some CPU cycles with an average value X_i^m . Hence, the average service rate (jobs/s) of incoming job requests of

Fig. 1: An example decentralized scenario where federated cloudlets from heterogeneous SPs can offload job requests to their neighbors but pay incentives only to the neighbors from different SPs.

class- m is defined as $\mu_i^m = \frac{F_i^m}{X_i^m}$. Therefore, μ_i^m indicates a parametric description of the job requests arriving at the i^{th} cloudlet. We consider that each i^{th} cloudlet has total $n_i \in \mathbb{Z}_{\geq 1}$ processors and dedicates $n_i^m \in \mathbb{R}_{\geq 1}$ processors for each job class- m . By using Google cluster-usage traces, the authors of [28] showed that *exponential distribution* fits perfectly with the service times of the job requests. In practice, some of the job requests can be completely parallelized and some cannot be parallelized at all. Hence, to guarantee the worst-case performance of the cloudlets, we model the cloudlets as $M/M/c$ queuing systems [29].

(c) QoS latency requirements of job requests: The individual job requests from mobile devices demand a certain number of CPU cycles to process the jobs within a predefined QoS latency target D_{Qm} [7]. However, in this paper we are considering a batch of incoming job requests rather than individual job requests to cloudlets. Thus, we denote the computational and latency requirements of the job requests of class- m to i^{th} cloudlet by the consolidated tuple $\langle \mu_i^m, \lambda_i^m, D_{Qm} \rangle$. To handle the processing of different job classes, we use *processor slicing* technique [22] to slice the total n_i processors into M slices of n_i^m processors for each job class- m . Moreover, each cloudlet uses a *timeslotted model* to ensure the QoS latency target D_{Qm} for the job requests of each job class-

Fig. 2: A schematic diagram to illustrate the design of timeslots.

m , as shown in Fig. 2. The duration of each fundamental timeslot is chosen to be equal to D_Q and without loss of generality, we assume that D_{Qm} values are integer multiples of D_Q , i.e., $D_{Q1} = I_1 \times D_Q$, $D_{Q2} = I_2 \times D_Q$, $D_{Q3} = I_3 \times D_Q$, and so on ($I_1, I_2, I_3, \dots, I_M$ are integer values). Depending on the stationarity of the incoming job request traffic, we choose a bigger time interval τ_x which is an integer multiple of all of $I_1, I_2, I_3, \dots, I_M$. Each cloudlet uses a prediction algorithm a few timeslots before the beginning of τ_x interval to predict the job request arrival rates for all the M job classes. Based on this predicted job request arrival rates, the processor slicing and the NE load balancing strategies for each cloudlet are computed. These values remain unchanged over each interval τ_x . As the job requests arrive within each timeslot, they are marked with an integer ($I_m - 1$). If some jobs could not be processed within that timeslot, they are rolled over to the next timeslot by decreasing the marking by 1. This can continue until those jobs are processed and the jobs can be deleted after the marking becomes 0. Therefore, the job request arrival queue of the cloudlets can maintain a steady-state unless they are extremely overloaded. As the $M/M/c$ queue provides the worst-case processing latency of the cloudlets, we are also ensuring that all the incoming job requests are processed when the average latency of each cloudlet is $\leq D_{Qm}$. Please refer to Appendix-A for more discussions on timeslots and modelling of cloudlets as $M/M/c$ queues.

(d) User mobility model: We assume that the mobile users cannot move beyond the coverage area of a cloudlet within a few milliseconds, thus consider the *quasi-static mobility model* for mobile users. This means that mobile users can be considered almost stationary to the corresponding cloudlets during computation offloading period, but may move on later [7]. Each cloudlet prioritizes the processing of the incoming job requests internally or offloads to a neighboring cloudlet to satisfy the QoS latency target D_{Qm} through some internal scheduling algorithm (beyond the scope of this paper).

III. ECONOMIC AND NON-COOPERATIVE LOAD BALANCING GAME AMONG CLOUDLETS

In this section, we formulate the load balancing problem among $N \geq 2$ neighboring federated cloudlets as a continuous-kernel non-cooperative game. In a practical deployment scenario, overloaded cloudlets intend to offload a fraction of its job requests to its under-loaded neighboring cloudlets. We denote the fraction of class- m job requests i^{th} cloudlet offloads to its j^{th} neighboring cloudlet by φ_{ij}^m . The complete job request offloading strategy space of all cloudlets for each job class- m is defined as a matrix $\Phi^m = ((\Phi_1^m)^T, (\Phi_2^m)^T, \dots, (\Phi_N^m)^T)^T \subset \mathbb{R}^{N \times N}$, where $\Phi_i^m = (\varphi_{i1}^m, \varphi_{i2}^m, \dots, \varphi_{iN}^m) \in \Phi_i^m \subset \mathbb{R}^N$, $\varphi_{ij} \in \Phi_{ij}^m = [0, 1] \subset \mathbb{R}$, $\sum_{j=1}^N \varphi_{ij}^m = 1, \forall i \in C$. In a stable market scenario, all the SPs tend to install cloudlets with similar processing capacity (i.e., $\mu_i^m = \mu_j^m, \forall i, j$) to meet a standard QoS for the same customer base. Hence, the total processing latency of the class- m job requests at i^{th} cloudlet with $\bar{\lambda}_i^m(\varphi_i^m, \varphi_{-i}^m) = \{(1 - \sum_{j \neq i} \varphi_{ij}^m) \lambda_i^m + \sum_{j \neq i} \varphi_{ji}^m \lambda_j^m\}$ and $\bar{\rho}_i^m = \bar{\lambda}_i^m / (n_i^m \mu_i^m)$ can be

derived as follows:

$$\mathbb{T}_i^m(\mu_i^m, \bar{\lambda}_i^m, n_i^m) = \frac{1}{\mu_i^m} + \frac{\mathcal{E}_C(n_i^m, \bar{\lambda}_i^m / \mu_i^m)}{n_i^m \mu_i^m - \bar{\lambda}_i^m}, \quad (1)$$

where, $\mathcal{E}_C(n_i^m, \bar{\lambda}_i^m / \mu_i^m)$ is the *Erlang-C formula*, given by:

$$\begin{aligned} \mathcal{E}_C(n_i^m, \bar{\lambda}_i^m / \mu_i^m) &= \frac{1}{1 + (1 - \rho_i^m) \left(\frac{n_i^m!}{(n_i^m \rho_i^m)^{n_i^m}} \right) \sum_{k=0}^{n_i^m-1} \frac{(n_i^m \rho_i^m)^k}{k!}} \\ &= \frac{1}{1 + \exp\left(\frac{\lambda_i^m}{\mu_i^m}\right) \left(n_i^m - \frac{\lambda_i^m}{\mu_i^m}\right) \left(\frac{\lambda_i^m}{\mu_i^m}\right)^{-n_i^m} \text{Gamma}\left(n_i^m, \frac{\lambda_i^m}{\mu_i^m}\right)}. \end{aligned} \quad (2)$$

Note that, we are using *incomplete Gamma function* in (2), which is often used for approximate and exact representation of many Mathematical series [30]. Each i^{th} cloudlet makes an optimal processor slicing by observing their load conditions and by solving the following optimization problem:

$$\begin{aligned} \mathcal{P}_i^m : \quad & \min_{(n_i^1, n_i^2, \dots, n_i^M)} \max_{m \in \mathcal{J}} \{t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m; n_i^m) - D_{Qm}\} \\ & \text{subject to} \quad 1 \leq n_i^m \leq n_i \forall m \in \mathcal{J}, \sum_{m=1}^M n_i^m = n_i. \end{aligned}$$

We consider that soft processor slicing is available and n_i^m can take any real and ≥ 1 value. Hence, $\mathcal{P}_i^m$ is a continuous convex optimization problem and can be solved by gradient projection algorithm (refer to Appendix-B).

A. Economic and Non-cooperative Game Formulation

In this paper, we consider the most commonly used pricing schemes e.g., *pay-as-you-go* policy, where users pay a fixed price per job request without any long-term commitments [31]. For the total amount of incoming class- m job requests from all the connected mobile devices, each i^{th} cloudlet earns a linearly proportional revenue ($\Omega_{i,1}^m$) per workload. Each i^{th} cloudlet also pays a linearly proportional price per workload ($\Omega_{ij,2}^m$) for offloading job requests to a neighboring j^{th} cloudlet from a different SP and also, receives a linearly proportional price for executing its neighbor's offloaded jobs. The cloudlets can also use cooperative or bargaining strategies among themselves to decide the value of $\Omega_{ij,2}^m$. We define a parameter γ_{ij} to distinguish the price for offloading a job request to neighboring cloudlets as follows:

$$\gamma_{ij} = \begin{cases} 1; & \text{if neighboring cloudlet belongs to different SP} \\ 0; & \text{if neighboring cloudlet belongs to the same SP} \end{cases}$$

This means that each i^{th} cloudlet pays a price to j^{th} cloudlet to offload any job requests when it belongs to another SP, i.e. $\gamma_{ij} = 1$. In addition to these, each i^{th} cloudlet pays a penalty price with a proportionality cost factor ($\Omega_{i,3}^m$) for exceeding the QoS target latency D_{Qm} . Note that if an overloaded cloudlet offloads some job requests to a neighboring cloudlet and it fails to process them for some reason, then the penalty is actually paid by the neighboring cloudlet. In this work, we consider a linear penalty price similar to the linear latency cost designed in [14]. Therefore, all the federated cloudlets with *utility functions* $\mathcal{U}_i^N(\varphi_i, \varphi_{-i}), \forall i \in C$, where

$\varphi_{-i} = (\varphi_1, \dots, \varphi_{i-1}, \varphi_{i+1}, \dots, \varphi_N)$, in the load balancing game intend to solve the following maximization problem:

$$\begin{aligned} \max_{\varphi_i \in \Phi_i} \mathcal{U}_i^N(\varphi_i, \varphi_{-i}) = & \sum_{m=1}^M \Omega_{i,1}^m \frac{\lambda_i^m}{n_i^m \mu_i^m} \\ & + \sum_{m=1}^M \sum_{j=1, j \neq i}^N \Omega_{ji,2}^m \gamma_{ji} \frac{\varphi_{ji}^m \lambda_j^m}{n_i^m \mu_i^m} - \sum_{m=1}^M \sum_{j=1, j \neq i}^N \Omega_{ij,2}^m \gamma_{ij} \frac{\varphi_{ij}^m \lambda_i^m}{n_j^m \mu_j^m} \\ & - \sum_{m=1}^M \Omega_{i,3}^m \left[\left(1 - \sum_{j=1, j \neq i}^N \varphi_{ij}^m \right) \frac{\lambda_i^m}{n_i^m \mu_i^m} \max \{0, (t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \bar{\lambda}_i^m) - D_{Qm})\} \right. \\ & \left. + \sum_{j=1, j \neq i}^N \frac{\varphi_{ji}^m \lambda_j^m}{n_i^m \mu_i^m} \max \{0, (t_{uj}^m + \mathbb{T}_i^m(\mu_i^m, \bar{\lambda}_i^m) + t_{ji}^m - D_{Qm})\} \right] \end{aligned} \quad (3)$$

$$\begin{aligned} \text{subject to } & 0 \leq \varphi_{ij}^m \leq 1, \forall m \in \mathcal{J} \\ & \sum_{m=1}^M b_i^m \varphi_{ij}^m \lambda_j^m \leq B_{ij}, \forall j \neq i \in C. \end{aligned} \quad (4)$$

The first term in (3) denotes the total payment received by the cloudlet from mobile users and is linearly proportional to the average workload. The second term denotes the payment i^{th} cloudlet receives from j^{th} cloudlet to execute its offloaded job requests and the third term denotes the payment i^{th} cloudlet makes to j^{th} cloudlet for offloading job requests. The fourth term denotes the penalty i^{th} cloudlet pays for overall latency (sum of transmission, processing, and queuing latencies) if it exceeds D_{Qm} against the total incoming class- m job requests, otherwise no penalty is applied. We denote the average round-trip data transmission latency among mobile devices and the corresponding i^{th} cloudlet by t_{ui} and the inter-cloudlet round-trip data transmission latency by t_{ij} , $\forall i, j \neq i \in C$. We also consider that overloaded cloudlets may face network bandwidth constraint while offloading job requests as $\sum_{m=1}^M b_i^m \varphi_{ij}^m \lambda_j^m \leq B_{ij}$, where b_i^m denotes the average number of bits/job request of class- m and B_{ij} denotes the bandwidth available in the link between i^{th} and j^{th} cloudlets. Each cloudlet needs to pay a price to network SPs for the bandwidth consumed by the offloaded job requests to neighboring cloudlets, but this price is paid separately and over a longer period of time. We assume that the cloudlets operate under the *condition of stable operation*, i.e., $\{(1 - \sum_{j \neq i} \varphi_{ij}^m) \lambda_i^m + \sum_{j \neq i} \varphi_{ji}^m \lambda_j^m\} / (n_i^m \mu_i^m) < 1, \forall i, j \neq i \in C, \forall m \in \mathcal{J}$. The utility of each cloudlet in this load balancing game is an affine function when the total latency is within D_{Qm} , otherwise, it becomes a non-linear function whose maximum value is achieved at total end-to-end latency equal to D_{Qm} . Hence, the cloudlets are always interested in gaining some incentives by receiving some extra job requests from neighboring cloudlets without exceeding D_{Qm} but, the utility starts to decrease beyond this point. Moreover, the *individual rationality* of each federated cloudlet is maintained due to the *default utility*, $\mathcal{U}_i^0, \forall i \in C$.

Furthermore, due to the utility function (3) and constraints (4), which does not provide an explicit latency bound on the participating cloudlets, even highly over-loaded cloudlets can participate in the game and can offload some of the job requests to the relatively under-loaded neighboring cloudlets. This leads to a utility higher than the utility achieved without participating in the game. Note that under such network load conditions, the game formulation in [14] that has explicit delay bound on participating cloudlets becomes infeasible and a valid NE solution can not be computed. We prefer to investigate the NE of the game Γ , because none of the federated

TABLE I: Symbol definitions

Symbol	Definition
N	Total number of federated cloudlets in the network
M	Total number of different job classes generated from mobile devices
μ_i^m	Average service rate of job class- m at i^{th} cloudlet
λ_i^m	Average job request arrival rate of job class- m to i^{th} cloudlet
$\bar{\lambda}_i^m$	Revealed average job request arrival rate of job class- m to i^{th} cloudlet
D_{Qm}	QoS latency target of job class- m
n_i	Total number of processors installed in i^{th} cloudlet
n_i^m	Number of processors for job class- m of i^{th} cloudlet
φ_{ij}^m	The fraction of jobs i^{th} cloudlet offloads to j^{th} cloudlet
$\Omega_{i,1}^m$	Revenue earned by i^{th} cloudlet per workload of class- m
$\Omega_{ji,2}^m$	Incentive paid for offloading class- m jobs by i^{th} to j^{th} cloudlet
$\Omega_{i,3}^m$	Penalty paid by i^{th} cloudlet for violating QoS target latency
$\Omega_{i,4}^m$	Penalty paid to the market regulator for not processing received jobs
t_{ui}^m	Data transmission latency between mobile devices and i^{th} cloudlet
t_{ij}^m	Data transmission latency between i^{th} and j^{th} cloudlets

cloudlets find it beneficial to deviate unilaterally from the NE computational offload strategy $\varphi^* = (\varphi_1^*, \varphi_2^*, \dots, \varphi_N^*)$.

Lemma 3.1. *The utility functions $\mathcal{U}_i^N(\varphi_i, \varphi_{-i}), \forall i \in C$ of the under-loaded cloudlets are affine and the overloaded cloudlets are quasi-concave functions of φ_i .*

Proof: Please refer to Appendix-C. ■

Theorem 3.2. *At least one pure strategy NE exists for the game $\Gamma = \langle C, (\Phi_i^m)_{i \in C, m \in \mathcal{J}}, (\mathcal{U}_i^N(\varphi_i, \varphi_{-i}))_{i \in C} \rangle$.*

Proof: In the game Γ , the strategy spaces of all the federated cloudlets Φ_i are *compact and convex* in nature. The utility functions $\mathcal{U}_i^N(\varphi_i, \varphi_{-i}), \forall i \in C$ are continuous functions of $(\varphi_i, \varphi_{-i})$ with the condition of stable operation, $\{(1 - \sum_{j \neq i} \varphi_{ij}^m) \lambda_i^m + \sum_{j \neq i} \varphi_{ji}^m \lambda_j^m\} / (n_i^m \mu_i^m) < 1, \forall i, j \neq i \in C, \forall m \in \mathcal{J}$. Moreover, we showed in Lemma 3.1 that $\mathcal{U}_i^N(\varphi_i, \varphi_{-i}), \forall i \in C$ are affine or quasi-concave functions of φ_i . These are the sufficient conditions to ensure the existence of a pure strategy NE for the non-cooperative load balancing game Γ . ■

B. Analysis of the Pure-Strategy Nash Equilibrium

We observe that the utility functions $\mathcal{U}_i^N(\varphi_i, \varphi_{-i}), \forall i \in C$ are non-differentiable in nature due to the presence of $\max\{0, x\}$ function. Hence, we cannot derive the *best response functions* of the cloudlets by directly differentiating the utility functions [32]. At first, we identify whether the cloudlets are under-loaded or overloaded and organize their utilities accordingly. As different job classes are processed independent of each other through processor slicing, each cloudlet can simultaneously be under-loaded for one job class while overloaded for some other job class. After this, we proceed to analyze the pure-strategy NE load balancing strategies. Intuitively, three cases can arise in practice and the pure strategy NE solutions are described as follows:

Case-1: $[t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m)] < D_{Qm}$, $[t_{uj}^m + \mathbb{T}_j^m(\mu_j^m, \lambda_j^m)] < D_{Qm}, \forall j \neq i \in C, \forall m \in \mathcal{J}$. In this case, all the cloudlets are under-loaded, i.e., they have sufficient computational resources to meet the QoS latency target D_{Qm} . Hence, the *unique NE solution* is $\varphi_{ij}^{m*} = \varphi_{ji}^{m*} = 0$. This implies that both the cloudlets can achieve their maximum utilities as the total revenue earned without offloading any job requests to each other.

Case-2: $[t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m)] \geq D_{Qm}$, $[t_{uj}^m + \mathbb{T}_j^m(\mu_j^m, \lambda_j^m)] \geq D_{Qm}, \forall j \neq i \in C, \forall m \in \mathcal{J}$. In this case, all the cloudlets are

overloaded and they can not reduce their individual latencies by offloading any job requests to each other. Thus, it is obvious that the *unique NE strategy* for both the cloudlets is not to offload any job requests to each other, i.e., $\varphi_{ij}^{m*} = \varphi_{ji}^{m*} = 0$.

Case-3: $[t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m)] \geq D_{Qm}$, $[t_{uj}^m + \mathbb{T}_j^m(\mu_j^m, \lambda_j^m)] < D_{Qm}$, $\forall i \in C_o, j \in C_u, m \in \mathcal{J}$, where $C_o \subset C$ is the set of overloaded cloudlets and $C_u = C \setminus C_o$ is the set of under-loaded cloudlets. We can mark the N_u under-loaded cloudlets and N_o overloaded cloudlets by observing λ_i^m such that $N_u + N_o = N$. Thus, j^{th} under-loaded cloudlets do not need to offload anything, i.e., $\varphi_{ji}^{m*} = 0$ but i^{th} cloudlet needs to offload their excess job requests to j^{th} cloudlets to meet the QoS target latency D_{Qm} , as long as overloaded cloudlets do not exceed D_{Qm} . Hence, the following *NE solution* is $\varphi_{ij}^{m*} > 0$ and $\varphi_{ji}^{m*} = 0$. As we can not solve this game analytically, we can verify this solution by a suitable algorithmic solution. As long as the under-loaded j^{th} cloudlets can process the entire extra load from i^{th} cloudlet, they accept the entire workload. However, when overloaded cloudlets cannot process the entire extra load, then they allow the i^{th} only to offload job requests partially such that they do not exceed their QoS target latency D_{Qm} . In this case, we realized that the NE load balancing strategies are not entirely controlled by the overloaded cloudlets but also by the under-loaded cloudlets. Hence, we introduce a new set of decision variables for each i^{th} cloudlet denoting the fraction of job requests received from all j^{th} cloudlets i.e., $\psi_i^m = (\psi_{1i}^m, \psi_{2i}^m, \dots, \psi_{Ni}^m)^T$ with the jointly shared equality constraints, $\varphi_{ji}^m = \psi_{ji}^m, \forall i, j \neq i \in C, m \in \mathcal{J}$. These new set of constraints play an essential role in the evaluation of NE load balancing strategies among federated cloudlets.

IV. DECENTRALIZED LOAD BALANCING FRAMEWORK

In a distributed cloudlet network, the incoming class- m job request arrival rate λ_i^m to each cloudlet $i \in C$ is known only to that cloudlet and no other entity in the network. We assume that the federated cloudlets are *non-cooperative* and *rational utility maximizers* and they do not share this *private information* with each other. Therefore, a reinforcement learning automata-based algorithm helps the cloudlets to independently make load balancing decisions only from their private information and to aid this decision-making process, we use the economic and non-cooperative game formulated in Section III. Moreover, some particular characteristics of the underlying game formulation is used to reduce the search space of the reinforcement learning algorithm and greatly improve the convergence rate of the algorithm. The fundamental stages of the overall control design are summarized below:

- (a) A *load-predictive learning algorithm* is executed by each cloudlet just before every time interval τ_{x-1} by using historical data to predict the incoming job request arrival rate of the next time interval τ_x . Each cloudlet uses their predicted job request arrival rates to perform processor slicing for the time interval τ_x .
- (b) Each cloudlet also estimates the transmission latency of the incoming job requests by using the given stochastic parameters of the access network between mobile devices

and cloudlets. Each cloudlet also estimates the intermediate transmission latencies with its neighboring cloudlets.

- (c) Using this learned information, each cloudlet shares a random amount of its job requests to the neighboring cloudlets, depending on the latest probability distribution over its strategy space and observes the utility and rewards received. Based on the reward values received at n^{th} and $(n-1)^{\text{th}}$ time-slots, each cloudlet updates the probability distribution over its strategy space for $(n+1)^{\text{th}}$ time-slot.

A. Distributed Reinforcement Learning Algorithm

In this subsection, we design a continuous-action reinforcement learning automata-based algorithm for learning the NE of the continuous-kernel non-cooperative load balancing game formulated in Section III. At first, we define the *mixed-strategy* of each i^{th} cloudlet as continuous probability density function (PDF) $f_i(\varphi_i)$ over its pure-strategy space Φ_i [33]. Therefore, the probability of randomly choosing an action within a close neighborhood of φ_{ij}^m by i^{th} cloudlet can be determined from the corresponding PDF $f_{ij}^m(\varphi_{ij}^m)$. The complete mixed-strategy of all the cloudlets is defined as $\mathcal{F} := f_1 \times \dots \times f_N$ over the complete pure-strategy space Φ . When each i^{th} cloudlet chooses an action φ_{ij}^m , i.e., offloads $\varphi_{ij}^m \lambda_i^m$ job requests to neighboring j^{th} cloudlets, then the *environment responds with a random reward* $\mathcal{R}_i(\varphi_i, \varphi_{-i}) \in [0, 1]$, which is defined as:

$$\mathcal{R}_i(\varphi_i, \varphi_{-i}) = \frac{\mathcal{U}_i^N(\lambda_i, \varphi_i, \varphi_{-i})}{\sum_{m=1}^M \max\{\Omega_{i,1}^m, \Omega_{ij,2}^m, \Omega_{i,3}^m\}}, \forall i \in C. \quad (5)$$

Note that constraint (4) is also considered while evaluating (5). In the load balancing game, with a continuous-action reinforcement learning automata-based algorithm, each i^{th} cloudlet starts with *uniform probability distributions* as their mixed-strategies over their individual pure-strategy action spaces and keeps on updating the PDFs based on the received rewards in the following time-slots to ultimately *find their pure-strategy NE* [34]. After exploring an action $\varphi_i^{(n)} \in \Phi_i$ during n^{th} time-slot, the PDFs are updated for $(n+1)^{\text{th}}$ time-slot by the following update rule:

$$f_i^{(n+1)}(\varphi_i) = \chi^{(n)} \left[f_i^{(n)}(\varphi_i) + \Theta^{(n)} (\mathcal{R}_i^{(n)} - \mathcal{R}_i^{(n-1)}) \exp \left\{ -\frac{1}{2} \left(\frac{\varphi_i - \varphi_i^{(n)}}{\sigma^{(n)}} \right)^2 \right\} \right], \quad (6)$$

where, Θ is the *learning rate parameter*, σ is the *spreading rate parameter* and χ is a *normalization factor* so that $\int_{-\infty}^{+\infty} f^{(n+1)} dz = 1$, for any z . Note that, our proposed reinforcement learning automaton (6) operates as *gradient bandit* algorithm, based on the idea of *stochastic gradient ascent* algorithms [35]. Moreover, the term $(\mathcal{R}_i^{(n)} - \mathcal{R}_i^{(n-1)})$ used in this model makes this algorithm *highly robust in tracking a non-stationary job request arrival process*. The PDFs are continuously updated by the cloudlets based on their private information and rewards received at every time-slot to learn the pure-strategy NE of the non-cooperative load balancing game and the space complexity of this algorithm is $O((N \times M)^2 \times L)$, where L is the length of memory required for storing the discrete version of $f_{ij}^m(\varphi_{ij}^m)$ [36].

Theorem 4.1. *The continuous-action reinforcement learning automata-based algorithm with update rule (6) converges to the pure-strategy Nash equilibrium of the non-cooperative load balancing game.*

Please refer to Appendix-D for the proof. Although, the convergence of the proposed algorithm is guaranteed but, we can speed up the convergence rate of the algorithm several times more by scaffolding our understanding about the underlying load balancing game. We observe that whenever some cloudlet is in under-loaded condition, i.e., $(t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m)) < D_{Qm}$, its NE strategy is not to offload any job requests to its neighboring cloudlets. Thus, all cloudlets can update their PDFs accordingly without exploring many job request offloading strategies as long as the under-load condition persists. In addition to this, during overload condition, i.e., $(t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m)) \geq D_{Qm}$, each cloudlet will offload only a portion of the received job requests and try to shift the peak of the PDF $f_{ij}^m(\varphi_{ij}^m)$ around the pure strategy NE solution φ_{ij}^{m*} , such that it can meet D_{Qm} with the rest of the job requests by itself. Hence, the corresponding search spaces can be reduced accordingly. As each cloudlet is unaware of the load condition of its neighboring cloudlets, some of the offloaded jobs may get dropped as the under-loaded cloudlet will process jobs as long as the QoS target latency is satisfied. Therefore, when the offloaded jobs are partially processed, each overloaded i^{th} cloudlet need to compute the fraction of processed job requests, denoted by $\hat{\psi}_{ij}^m$. This implies that when i^{th} cloudlet offloads $\varphi_{ij}^m \lambda_i^m$ job requests but the j^{th} cloudlet processes only $\hat{\psi}_{ij}^m \lambda_i^m$ jobs, where $0 \leq \hat{\psi}_{ij}^m < \varphi_{ij}^m$. Note that cloudlets do not get incentives for dropping offloaded job requests and hence, the rational cloudlets do not drop job requests intentionally. In such cases, the i^{th} cloudlet updates the PDF by using $\hat{\psi}_{ij}^m$ in (6) instead

Algorithm 1 Distributed Reinforcement Learning Automata-based Algorithm for learning NE of the Load Balancing Game

- 1: **Initialization:** Set the iteration index $n = 0$ and the PDFs $f_{ij}^{m,(n)}(\varphi_{ij}^m) = \text{uniform}(\Phi_{ij}^m), \forall i, j \in \mathcal{C}, m \in \mathcal{J}$.
- 2: **Output:** The pure-strategy NE of the non-cooperative load balancing game.
- 3: **if** i^{th} cloudlet is under-loaded, i.e., $(t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m)) < D_{Qm}$ **then** choose not to offload, i.e., $\varphi_i^{m,(n)} = \mathbf{0}$;
- 4: **else** i^{th} cloudlet randomly choose an action $\varphi_i^{(n)}$ based on its latest mixed-strategy $f_i^{(n)}(\varphi_i)$.
- 5: **if** j^{th} cloudlet is overloaded or partially processes the jobs received from all i^{th} cloudlets **then** all i^{th} cloudlets compute that only $\hat{\psi}_{ij}^m \lambda_i^m$ jobs are processed ($0 \leq \hat{\psi}_{ij}^m < \varphi_{ij}^m$), where $\hat{\psi}_{ij}^m \lambda_i^m$ are chosen according to the ratio of $\varphi_{ij}^m \lambda_i^m$.
- 6: At the end of n^{th} time-slot, each i^{th} cloudlet receives a reward $\mathcal{R}_i^{(n)}$ from the environment.
- 7: Each i^{th} cloudlet update their mixed-strategy $f_i^{(n+1)}(\varphi_i)$ by the reinforcement learning automaton for $\varphi_i^{(n)} \in \Phi_i$:

$$f_i^{(n+1)}(\varphi_i) = \chi^{(n)} \left[f_i^{(n)}(\varphi_i) + \Theta^{(n)} \left(\mathcal{R}_i^{(n)} - \mathcal{R}_i^{(n-1)} \right) \exp \left\{ -\frac{1}{2} \left(\frac{\varphi_i - \varphi_i^{(n)}}{\sigma^{(n)}} \right)^2 \right\} \right]$$

- 8: Set $n \leftarrow n + 1$; go to Step 3.

(a) Average end-to-end latency of cloudlets with $n_i^m \mu_i^m = 1000$ jobs/s.

(b) Average utility values of cloudlets with $n_i^m \mu_i^m = 1000$ jobs/s.

Fig. 3: Comparison of performance between our proposed game and other games proposed in [14], [16] in terms of end-to-end latency and utility values with a difference of [0-200] jobs/s among incoming job request arrival rates of neighboring cloudlets.

of φ_{ij}^m . Furthermore, when i^{th} cloudlet is under-loaded and receives job requests from multiple cloudlets but can partially process the job requests i.e., $(t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m)) < D_{Qm}$ but $(t_{uj}^m + \mathbb{T}_i^m(\mu_i^m, (\lambda_i^m + \sum_{j \neq i} \varphi_{ji}^m \lambda_j^m)) + t_{ji}^m) \geq D_{Qm}$ the residual job processing capacity of the i^{th} cloudlet is distributed according to the ratio of $\varphi_{ji}^m \lambda_j^m$. The proposed algorithm is summarized in Algorithm 1.

V. RESULTS AND DISCUSSIONS

In this section, we investigate various fundamental properties of the proposed load balancing strategy through numerical evaluations. For this purpose, we consider a set of 10 neighboring federated cloudlets. At first, we consider a single job class- m to compare system performance with a few existing frameworks and the average processing rate of each of the cloudlets with multiple processors as $n_i^m \mu_i^m = 1000$ jobs/s and incoming job request rates to each cloudlet λ_i^m varies within 0-1500 jobs/s. We consider the duration of each timeslot as $D_Q = 5$ msec, the average value of latency between mobile users and cloudlets t_{ui}^m as 2 msec, the intermediate transmission latency between neighboring cloudlets t_{ij}^m varies within 0.5-1 msec, the number of bits/job request b_{ij}^m varies within 100-200 KB [37], and the available bandwidth B_{ij} lies within 0.5-1 Gbps [1]. The optimal values of proportionality price factors $\Omega_{i,1}^m$, $\Omega_{ij,2}^m$, $\Omega_{i,3}^m$, and $\Omega_{i,4}^m$ can be determined by studying the market equilibrium conditions for providing

Fig. 4: Comparison of actual (learned) and simulated NE utility values and the average end-to-end latencies in a three cloudlet and two job classes scenario. Cloudlets use fair processor slicing method for handling the processing of different job classes.

cloud-based services [38]. In actual practice, sometimes the proper price factors are also determined by applying the *multiple criteria decision-making theory* [39]. However, in this work we arbitrarily choose $\Omega_{i,1}^m = 5 \times 10^3$, $\Omega_{ij,2}^m = 3 \times 10^4$, and $\Omega_{i,3}^m = 9 \times 10^4$ cost/unit load such that feasible solutions corresponding to aforementioned network scenario can be obtained.

In Fig. 3(a), we compare average end-to-end latency of all the participating cloudlets against job request arrival rate with our currently proposed framework and the frameworks proposed in [14] (labeled as “ref. game-1”) and [16] (labeled as “ref. game-2”), respectively. To make a fair comparison, we consider only a single class of job requests with QoS latency target $D_{Q1} = 10$ msec (as [14] and [16] deal with a single job class), the difference in job request arrival rates among the under and overloaded cloudlets is within 0-200 jobs/s, and the service rates of all the cloudlets are $n_i^m \mu_i^m = 1000$ jobs/s. The characteristics of our proposed game-theoretic load balancing framework are intuitively explained in Appendix-C and we see that all the frameworks yield almost similar average latency under *low and medium load conditions* and the overall latency is within QoS target latency (observe the red dashed line). Hence, the users do not experience any degradation in QoS and latency minimization does not serve any significant impact.

Nonetheless, under *high load conditions*, when all the cloudlets become sufficiently overloaded, ref. game-1 becomes infeasible as some of the cloudlets start to violate explicit

latency constraints and ref. game-2 appears to overload the under-loaded cloudlets by uncontrolled offloading of the job requests. However, our framework still allows the overloaded cloudlets to strategically offload (partial offload) some job requests to relatively under-loaded cloudlets that can still accept some job requests while meeting the QoS latency target $D_{Q1} = 10$ msec. Therefore, the overall latency performance becomes comparatively better than ref. game-1 and ref. game-2. Note that neither of the ref. game-1 and ref. game-2 considers a bandwidth constraint on the inter-cloudlet links which is considered in our framework. Therefore, if the bandwidth is insufficient on the inter-cloudlet links, the performance of ref. game-1 and ref. game-2 will be much worse. However, for the sake of a fair comparison, we have considered that the bandwidth of inter-cloudlet links is sufficient for offloading excess job requests of overloaded cloudlets. Next, Fig. 3(b) shows a comparison among average utility values of all the participating cloudlets against job request arrival rate. As economic incentives are not involved in ref. game-1, and ref. game-2 frameworks, we first find their respective NE load balancing strategies and then compute the inter-cloudlet payment for offloading job requests and penalties for violating QoS latency target by our considered parameters. This brings all the load balancing frameworks on a common ground against federated cloudlet scenarios. Under *low load conditions*, all the load balancing frameworks yield similar utilities because none of the cloudlets offload any job requests. However, it is clear

(a) Average NE utility learning accuracy with $\Theta = 0.9$ and $\sigma = 0.03$.

(b) Average NE utility learning accuracy with $\Theta = 4.0$ and $\sigma = 0.01$.

Fig. 5: Average NE utility learning accuracy of Algorithm 1 for load balancing game among three cloudlets with dynamically varying job request arrival rates (λ_i^m) to all cloudlets.

that the average economic utility values of the cloudlets with our game formulation are comparatively better than both ref. game-1 and ref. game-2, especially under medium and high-load conditions.

To observe the performance of our proposed load balancing game with some real-world traces, we consider the cluster usage trace released by Google in 2011 [40]. We have used a dataset of a total size of nearly 10^6 samples for this purpose. We identify two different classes of incoming job requests with $D_{Q1} = 10$ msec and $D_{Q2} = 20$ msec and fit the PDFs of inter-arrival times with exponential PDFs (refer to Appendix-A). We perform an event-driven simulation on OMNeT++ with these traces among three cloudlets. Each of these cloudlets has $n_i = 10$ processors and the different number of processors dedicated to the two different job classes (n_i^m) are decided by solving $\mathcal{P}_i^m$. The job request service rates are $\mu_i^1 = 250$ jobs/s and $\mu_i^2 = 200$ jobs/s and the job request arrival rates (λ_i^m) for both the classes vary within 0-1500 jobs/s. The intermediate transmission latencies and bandwidths are $t_{12} = t_{21} = 0.5$ msec, $t_{23} = t_{32} = 0.7$ msec, $t_{31} = t_{13} = 0.9$ msec, $B_{12} = B_{21} = 0.6$ Gbps, $B_{23} = B_{32} = 0.8$ Gbps, and $B_{13} = B_{31} = 1$ Gbps, respectively. The duration of each of the utility evaluation time-interval τ_x is 1 sec. Each cloudlet uses LSTM networks to predict their incoming job request arrival rates one timeslot before the beginning of each time interval τ_x and computes the optimal processor slicing by solving $\mathcal{P}_i^m$. Then based on the reinforcement learning automata, each overloaded cloudlet

(a) Average NE learning accuracy with $\sigma = 0.01$.

(b) Average NE learning accuracy with $\Theta = 0.9$.

Fig. 6: Average NE learning accuracy of Algorithm 1 against stationarity time of job request arrival rates (λ_i^m) to all cloudlets with variations of learning rate Θ and spreading rate σ .

randomly offloads a fraction of their incoming job requests to their under-loaded neighbors during all intermediate timeslots within τ_x interval. Every time a job request arrives, each i^{th} overloaded cloudlet randomly offloads the job requests to its j^{th} neighboring cloudlet as per φ_{ij}^{m*} and the job processing simulation is implemented in the OMNeT++ environment as described in Section II.

In Fig. 4(a), we show three subplots to compare the actual and simulated utilities of each of the cloudlets. The actual utilities are referred to the final steady-state utility values obtained by the reinforcement learning algorithm against each set of λ_i^m values of all the cloudlets. Firstly, we plot the actual utilities with the actual job request arrival rates. Secondly, we plot the simulated utilities with actual job request arrival, but the NE solution is computed by using the predicted job request arrival rates. Hence, the NE solution is erroneous and the simulated utilities also deviate from the learned utilities. We observed that the simulated utilities have a mean error of 10% from the actual utilities. Out of this, nearly 8% error was due to the prediction error in job request arrival rates and the rest of the error was due to the approximation error in modeling the job request arrival process as a Poisson process. In Fig. 4(b), we show the actual and simulated average latencies of each of the cloudlets. As *fair processor slicing* method is employed, processors among all the job classes are shared fairly. This implies that if the job request arrival rate for one job class becomes very high, it gets a higher number of processors to lower down its processing latency, but the other classes are not

deprived so that their processing latencies become very high in turn. Thus, if a cloudlet appears to be overloaded even after internal processor slicing, it should try to offload its extra job requests to its neighbors to meet the respective D_{Qm} values.

Again, we consider the same scenario of three federated cloudlets with $n_1^m \mu_1^m = n_2^m \mu_2^m = n_3^m \mu_3^m = 1000$ jobs/s and $D_{Qm} = 10$ msec. Moreover, the job request arrival rates λ_i^m to each of these cloudlets are dynamically varying and the values are considered from the Google cluster traces. Therefore, the utilities of each i^{th} cloudlet $\mathcal{U}_i^N$ also changes accordingly. Each cloudlet predicts their incoming job request arrival rates one-timeslot before the beginning of τ_x time interval with an LSTM network, trained by a stochastic sub-gradient algorithm to decide whether it is under-loaded or overloaded. If a cloudlet finds itself overloaded, it starts to randomly offload its extra load to under-loaded neighboring cloudlets according to its reinforcement learning automata. Fig. 5(a) has a learning rate $\Theta = 0.9$ and spreading rate $\sigma = 0.03$ and Algorithm 1 takes some time to learn the actual utility values due to more exploration. Fig. 5(b) has $\Theta = 4.0$ and $\sigma = 0.01$ where Algorithm 1 learns the actual utility values relatively faster but performance may degrade during sudden changes due to less exploration. Note that if a cloudlet is under-loaded then it decides its NE strategy without any exploration. Overall, we observe that by employing Algorithm 1, the cloudlets can achieve fairly accurate utility values.

From the previous results, we found that it is essential to choose the learning rate and spreading rate parameters in such a way so that a proper balance between exploration and exploitation is maintained against the stationarity time of job request arrival rates. Thus, in Fig. 6 we plot the average NE learning accuracy against the stationarity period of the job request arrival rate. We vary the stationarity time from 0.5 sec to 150 sec and also tune Θ from 0.5 to 3 with $\sigma = 0.01$ in Fig. 6(a), and tune σ from 0.009 to 0.030 with $\Theta = 0.9$ in Fig. 6(b). We observe in general, that the performance of our proposed Algorithm 1 increases as the stationarity of the job request arrival rate increases because the algorithm is given more time-slots to exploit and explore the search space.

VI. CONCLUSIONS

In this paper, we have proposed a novel load balancing framework among federated neighboring cloudlets from the same as well as different SPs. In this framework, a distributed continuous-action reinforcement learning automata-based algorithm is used to facilitate the neighboring cloudlets to find their load balancing strategies. The primary advantage of this algorithm is that a minimal exchange of control messages among neighboring cloudlets is required. We formulate the load balancing problem among multiple federated cloudlets as an economic and non-cooperative game where we focus on latency bound rather than latency minimization. Furthermore, this framework acts like a non-cooperative game among neighboring cloudlets from different SPs and like an optimization problem among cloudlets from the same SP. We use some typical characteristics of this underlying game to achieve faster convergence for the proposed reinforcement

learning automata. In addition, this load balancing framework is capable of handling job requests from heterogeneous classes by employing processor slicing, which is unique over state-of-the-art frameworks. Through numerical evaluations, we have also shown that our proposed framework achieves better economic utilities under medium and high load conditions compared to some existing frameworks. We have used real job request arrival traces in an event-driven simulation for performance evaluation of the load balancing framework in a realistic scenario. We showed that the average accuracy of the utility values of the cloudlets with predicted job request arrival rates is nearly 90% of the utility values with actual job request arrival rates. A large percentage of the error arises from the job request arrival rate prediction error and only a very minimal percentage from our proposed model.

REFERENCES

- [1] E. Wong, M. P. I. Dias, and L. Ruan, "Predictive Resource Allocation for Tactile Internet Capable Passive Optical LANs," *Journal of Lightwave Technology*, vol. 35, no. 13, pp. 2629–2641, July 2017.
- [2] S. Mondal, L. Ruan, M. Maier, D. Larrabeiti, G. Das, and E. Wong, "Enabling Remote Human-to-Machine Applications With AI-Enhanced Servers Over Access Networks," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 889–899, 2020.
- [3] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Nakanlahiji, J. Kong, and J. P. Jue, "All One Needs to Know About Fog Computing and Related Edge Computing Paradigms: A Complete Survey," *Journal of Systems Architecture*, 2019.
- [4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.
- [5] X. Meng, W. Wang, and Z. Zhang, "Delay-Constrained Hybrid Computation Offloading With Cloud and Fog Computing," *IEEE Access*, vol. 5, pp. 21 355–21 367, 2017.
- [6] S. Ahn, J. Lee, S. Park, S. H. S. Newaz, and J. K. Choi, "Competitive Partial Computation Offloading for Maximizing Energy Efficiency in Mobile Cloud Computing," *IEEE Access*, vol. 6, pp. 899–912, 2018.
- [7] H. Cao and J. Cai, "Distributed Multiuser Computation Offloading for Cloudlet-Based Mobile Cloud Computing: A Game-Theoretic Machine Learning Approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 752–764, Jan 2018.
- [8] D. T. Nguyen, L. B. Le, and V. Bhargava, "Price-based Resource Allocation for Edge Computing: A Market Equilibrium Approach," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2018.
- [9] A. G. Tasiopoulos, O. Ascigil, I. Psaras, S. Tournis, and G. Pavlou, "On-path Cloudlet Pricing for Low Latency Application Provisioning," in *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, June 2018, pp. 31–36.
- [10] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, October 2016.
- [11] F. Zhang and M. M. Wang, "Stochastic Congestion Game for Load Balancing in Mobile Edge Computing," *IEEE Internet of Things Journal*, pp. 1–1, 2020.
- [12] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet Load Balancing in Wireless Metropolitan Area Networks," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.
- [13] L. Liu, S. Chan, G. Han, M. Guizani, and M. Bandai, "Performance Modeling of Representative Load Sharing Schemes for Clustered Servers in Multiaccess Edge Computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4880–4888, June 2019.
- [14] L. Chen, S. Zhou, and J. Xu, "Computation Peer Offloading for Energy-Constrained Mobile Edge Computing in Small-Cell Networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, Aug 2018.
- [15] S. Mondal, G. Das, and E. Wong, "Computation Offloading in Optical Access Cloudlet Networks: A Game-Theoretic Approach," *IEEE Communications Letters*, vol. 21, no. 7, pp. 1481–1484, July 2018.

- [16] C. Liu, K. Li, and K. Li, "A Game Approach to Multi-servers Load Balancing with Load-Dependent Server Availability Consideration," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2018.
- [17] S. Mondal, G. Das, and E. Wong, "A Game-Theoretic Approach for Non-Cooperative Load Balancing Among Competing Cloudlets," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 226–241, 2020.
- [18] Y. Jiang, "A Survey of Task Allocation and Load Balancing in Distributed Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 585–599, Feb 2016.
- [19] K. D. Hoang, C. Wayllace, W. Yeoh, J. Beal, Soura, Dasgupta, Y. Mo, A. M. Paulos, and J. Schewe, "New Distributed Constraint Satisfaction Algorithms for Load Balancing in Edge Computing : A Feasibility Study," in *10th International Workshop on Optimization in Multiagent Systems (OptMAS)*, May 2019.
- [20] X. Zhou, K. Wang, W. Jia, and M. Guo, "Reinforcement learning-based adaptive resource management of differentiated services in geo-distributed data centers," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, June 2017, pp. 1–6.
- [21] J. Baek, G. Kaddoum, S. Garg, K. Kaur, and V. Gravel, "Managing Fog Networks using Reinforcement Learning Based Load Balancing Algorithm," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–7.
- [22] J. Baek and G. Kaddoum, "Heterogeneous Task Offloading and Resource Allocations via Deep Recurrent Reinforcement Learning in Partial Observable Multi-Fog Networks," *IEEE Internet of Things Journal*, pp. 1–1, 2020.
- [23] S. Penmatsa and A. T. Chronopoulos, "Game-theoretic Static Load Balancing for Distributed Systems," *Journal of Parallel and Distributed Computing*, vol. 71, no. 4, pp. 537 – 555, 2011.
- [24] W. Jiang, B. Han, M. A. Habibi, and H. D. Schotten, "The Road Towards 6G: A Comprehensive Survey," *IEEE Open Journal of the Communications Society*, vol. 2, pp. 334–366, 2021.
- [25] A. Schaerf, Y. Shoham, and M. Tennenholtz, "Adaptive Load Balancing: A Study in Multi-agent Learning," *Journal of Artificial Intelligence Research*, vol. 2, no. 1, pp. 475–500, May 1995.
- [26] S. Chen, M. Ghorbani, Y. Wang, P. Bogdan, and M. Pedram, "Trace-Based Analysis and Prediction of Cloud Computing User Behavior Using the Fractal Modeling Technique," in *2014 IEEE International Congress on Big Data*, 2014, pp. 733–739.
- [27] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997.
- [28] C. Jiang, Y. Chen, Q. Wang, and K. J. R. Liu, "Data-Driven Stochastic Scheduling and Dynamic Auction in IaaS," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1–6.
- [29] M. Jia, J. Cao, and W. Liang, "Optimal Cloudlet Placement and User to Cloudlet Allocation in Wireless Metropolitan Area Networks," *IEEE Transactions on Cloud Computing*, vol. 5, no. 4, pp. 725–737, Oct 2017.
- [30] P. Amore, "Asymptotic and exact series representations for the incomplete Gamma function," *Europhysics Letters (EPL)*, vol. 71, no. 1, pp. 1–7, jul 2005.
- [31] IBM Cloud pricing. [Online]. Available: <https://www.ibm.com/cloud/pricing>
- [32] N. McCarty and A. Meirowitz, *Political Game Theory: An Introduction*, ser. Analytical Methods for Social Research. Cambridge University Press, 2007.
- [33] Y. Narahari, *Game Theory and Mechanism Design*. World Scientific Publishing Company Pte. Limited, 2014.
- [34] A. Rodríguez, P. Vrancx, R. Grau, and A. Nowé, "A Reinforcement Learning Approach to Coordinate Exploration with Limited Communication in Continuous Action Games," *The Knowledge Engineering Review*, vol. 31, no. 1, p. 77–95, 2016.
- [35] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.
- [36] V. V. Phansalkar, P. S. Sastry, and M. A. L. Thathachar, "Absolutely Expedient Algorithms for Learning Nash Equilibria," *Proceedings Mathematical Sciences*, vol. 104, no. 1, pp. 279–294, Feb 1994.
- [37] S. Mondal, G. Das, and E. Wong, "Cost-Optimal Cloudlet Placement Frameworks over Fiber-Wireless Access Networks for Low-Latency Applications," *Journal of Network and Computer Applications*, vol. 138, pp. 27 – 38, 2019.
- [38] Y. Feng, B. Li, and B. Li, "Price Competition in an Oligopoly Market with Multiple IaaS Cloud Providers," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 59–73, Jan 2014.
- [39] J. Wallenius, P. C. Fishburn, S. Zionts, J. S. Dyer, R. E. Steuer, and K. Deb, "Multiple Criteria Decision Making, Multiattribute Utility Theory: Recent Accomplishments and What Lies Ahead," *Management Science*, vol. 54, no. 7, pp. 1336–1349, 2008.
- [40] Google clusterdata 2011 traces. [Online]. Available: https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md

Sourav Mondal received the B.Tech degree in Electronics and Communication Engineering from Kalyani Govt. Engineering College, affiliated to West Bengal University of Technology and the M.Tech in Telecommunication Systems Engineering from the Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology Kharagpur in 2012 and 2014, respectively. He has worked in Qualcomm India Pvt. Ltd., Hyderabad from 2014 to 2016. Currently, he has joined Department of Electrical and Electronic Engineering of the University of Melbourne to pursue his doctoral studies and has started to work on Cloudlet based Edge-Computing for low-latency applications over optical access networks.

well as wireless networking.

Goutam Das has obtained his M.Tech. degree from the Indian Institute of Technology, Kharagpur (India) in 2001, and his Ph.D. degree from the University of Melbourne (Australia) in 2008. He has also worked as a senior research engineer in General Electric R&D center for 3.5 years from the year 2001 to 2004. He has also worked as a post-doctoral fellow at Ghent University (Belgium) from 2009 to 2011. Currently he is working as an Assistant Professor in the Indian Institute of Technology, Kharagpur. His research interest is in the area of both optical as

board of the Journal of Lightwave Technology and the Journal of Optical Communications and Networking.

Elaine Wong received B.E. and Ph.D. degrees from the University of Melbourne, Melbourne, Victoria, Australia. She is currently a Professor at the University of Melbourne. Her research interests include energy-efficient optical and wireless networks, optical-wireless integration, broadband applications of vertical-cavity surface-emitting lasers, wireless sensor body area networks, and emerging optical and wireless technologies for Tactile Internet. She has coauthored more than 150 journal and conference publications. She has served on the editorial

Appendices for A Decentralized Economic Load-Balancing Framework among Federated Cloudlets

Sourav Mondal, *Member, IEEE*, Goutam Das, *Member, IEEE*, and Elaine Wong, *Senior Member, IEEE*

APPENDIX A

DETAILS OF TIMESLOTS TO GUARANTEE QoS LATENCY AND $M/M/c$ QUEUING MODEL FOR CLOUDLETS

(a) Design of timeslots to guarantee QoS latency: In our proposed load balancing game, each cloudlet uses a *timeslotted model* to ensure the QoS latency target D_{Qm} for the job requests of each job class- m . The duration of each fundamental timeslot is chosen to be equal to D_Q and without loss of generality, we assume that D_{Qm} values are integer multiples of D_Q , i.e., $D_{Q1} = I_1 \times D_Q$, $D_{Q2} = I_2 \times D_Q$, $D_{Q3} = I_3 \times D_Q$, and so on ($I_1, I_2, I_3, \dots, I_M$ are integer values). Depending on the stationarity of the incoming job request traffic, we choose a bigger time interval τ_x which is an integer multiple of all of $I_1, I_2, I_3, \dots, I_M$. Each cloudlet uses a workload predictive algorithm a few timeslots before the beginning of each τ_x time interval to predict the job request arrival rates for all the M job classes. Based on this predicted job request arrival rates, each cloudlet can solve $\mathcal{P}_i^m$ to decide the processor slicing. They can use this information to calculate the pure strategy Nash equilibrium (NE) load balancing strategies. This processor slicing and the NE load balancing strategies are applied when the time interval τ_x actually begins. During each τ_x interval, the processor slicing and the NE load balancing strategies remain unchanged over all the timeslots.

The cloudlets need to use the timeslotted model to ensure the target QoS latency for each job request class. The job requests arrive continuously and randomly within each timeslot and are marked with an integer $\lfloor \frac{D_{Qm} - D_Q}{D_Q} \rfloor = (I_m - 1)$. The duration of a timeslot D_Q is subtracted from D_{Qm} to adjust the time lost due to the transmission latencies. If some jobs could not be processed within the timeslot of job request arrival, especially the jobs arriving at the verge of finishing of the timeslot, they are rolled over to the next timeslot by decreasing the marking by 1. This process can continue until those jobs are processed and the jobs can be deleted after the marking becomes 0. This process is followed even if the ongoing τ_x time interval is over and the next time slot τ_{x+1} begins. Therefore, the job request arrival queue of the cloudlets can maintain a steady-state unless they are extremely overloaded. As the job request arrival rates are predicted slightly before the actual job request arrival, all these network management operations can be done very efficiently.

For more clarity on this timeslotted protocol design, we provide the following numerical example. We consider the values $D_Q = 1$ msec, $D_{Q1} = 5$ msec, $D_{Q2} = 10$ msec, and

$\tau_x = 1$ sec. Thus, there are 1000 timeslots of $D_Q = 1$ msec within each interval of $\tau_x = 1$ sec. Say, 5 timeslots before the beginning of each τ_x interval, the processor slicing and NE load balancing strategies are computed based on the one timeslot ahead predicted job request arrival rates. When job requests of class-1 arrive within a particular timeslot, they are marked as 4. If some of these jobs could not be processed within that timeslot, they are rolled over to the next timeslot while updating the marking as 3 and continued upto 0. If these jobs are still unprocessed after this timeslot, then they are dropped. Similarly, when job requests of class-2 arrive within a particular timeslot, they are marked as 9 and the same process is followed. At the end of each τ_x interval, the processor slicing and NE load balancing strategies are re-calculated. Therefore, the job request arrival queue of the cloudlets can always remain in the steady-state.

(b) $M/M/c$ queuing model for cloudlets: We consider that each cloudlet has a finite number of processors, say c . In practice, some of the jobs can be completely parallelized and some cannot be parallelized at all. Hence, the actual performance of each cloudlet will lie within the performance bounds provided by $M/M/c$ and $M/M/1$ queuing systems. Recall that the average processing latency of an $M/M/c$ queuing system with job request arrival rate $= \lambda$ jobs/s, service rate $= \mu$ jobs/s and $\rho = \lambda/(c\mu)$ is expressed as follows [1]:

$$\mathbb{E}(T) = \frac{1}{\mu} + \frac{\mathcal{E}_C(c, \lambda/\mu)}{c\mu - \lambda}, \quad (1)$$

where, $\mathcal{E}_C(c, \lambda/\mu)$ is the *Erlang-C formula*, given by:

$$\mathcal{E}_C(c, \lambda/\mu) = \frac{1}{1 + (1 - \rho) \left(\frac{c!}{(c\rho)^c} \right) \sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!}}. \quad (2)$$

Again, the average processing latency of an $M/M/1$ queuing system with the combined service rate of $c\mu$ jobs/s is expressed as follows:

$$\mathbb{E}(\hat{T}) = \frac{1}{c\mu - \lambda} = \frac{1}{\mu} + \frac{\rho}{c\mu - \lambda}. \quad (3)$$

Therefore, under *light load conditions*, we have $\rho \ll 1$, $\frac{\rho}{c\mu - \lambda} \approx 0$, $\frac{\mathcal{E}_C(c, \lambda/\mu)}{c\mu - \lambda} \approx 0$ and hence derive the following:

$$\frac{\mathbb{E}(T)}{\mathbb{E}(\hat{T})} = \frac{\frac{1}{\mu} + \frac{\mathcal{E}_C(c, \lambda/\mu)}{c\mu - \lambda}}{\frac{1}{\mu} + \frac{\rho}{c\mu - \lambda}} \approx \frac{\frac{1}{\mu}}{\frac{1}{\mu}} \approx c. \quad (4)$$

Fig. 1: $M/M/2$ queuing system with service rate = 1 jobs/sec and an $M/M/1$ queuing system with service rate = 2 jobs/sec.

On the other hand, under *heavy load conditions*, we have $\rho \approx 1$, $\mathcal{E}_C(c, \lambda/\mu) \approx 1$, $\frac{1}{\mu} \ll \frac{\rho}{c\mu - \lambda}$, $\frac{1}{\mu} \ll \frac{\mathcal{E}_C(c, \lambda/\mu)}{c\mu - \lambda}$ and hence derive the following:

$$\frac{\mathbb{E}(T)}{\mathbb{E}(\hat{T})} = \frac{\frac{1}{\mu} + \frac{\mathcal{E}_C(c, \lambda/\mu)}{c\mu - \lambda}}{\frac{1}{\mu} + \frac{\rho}{c\mu - \lambda}} \approx \frac{\frac{1}{c\mu - \lambda}}{\frac{1}{c\mu - \lambda}} \approx 1. \quad (5)$$

From the above analysis, we observe that $\mathbb{E}(T) \geq \mathbb{E}(\hat{T})$ holds true under low as well as high load conditions. Therefore, to guarantee the worst-case performance of the cloudlets in terms of the processing latency, we model the cloudlets as $M/M/c$ queuing systems. To illustrate this, we compare the performance of an $M/M/2$ queuing system with service rate = 1 jobs/sec for each of the processors and an $M/M/1$ queuing system with service rate = 2 jobs/sec in Fig. 1. Hence, we have modeled the cloudlets as $M/M/c$ queuing systems in our manuscript as this guarantees the worst-case average processing latency of the cloudlets.

(c) Poisson job request arrival to cloudlets: To validate our assumption that the incoming job requests to the cloudlets follow the Poisson process, we consider the cluster usage trace released by Google in 2011 [2]. The Fig. 2a shows the inter-arrival times of the job requests from two different job classes, i.e., the difference between the timestamps of the consecutive job requests. From this plot, we can observe two primary characteristics, viz., *self-similarity* and *non-stationarity*. These properties of the Google cluster traces were studied in more details by the authors of [3]. Due to self-similarity property, the series of job request inter-arrival times looks the same under any magnification of scales. The non-stationarity property indicates that the distribution and statistical moments of the job request inter-arrival times vary over time. As the job request arrival process is a non-stationary random process, we considered small windows of 1 sec and attempted to fit the distribution of the inter-arrival times with exponential probability distribution in Fig. 2b. We checked that the distribution fitting satisfies a significance level of 0.05 (verified with Chi-squared goodness-of-fit test). Similar observations were also reported by the authors of [4]. Note that we have scaled up the job request arrival rate (jobs per 300 sec are treated as jobs

per 1 sec) since we found that the job request arrival rates to the Google clusters are much lower than our expected job request arrival rates to the cloudlets.

In Fig. 3a, we show job request arrival rates for two different job classes from the Google cluster traces. As the job request arrival process is self-similar and non-stationary, each cloudlet uses long short-term memory (LSTM) networks and stochastic gradient descent algorithm to predict the incoming job request arrival rates to the cloudlets. In this figure, the job request arrival rates over the first 100 sec interval are used to train the LSTM network. Once the network is trained, it is used to forecast the job request arrival rates 1 sec before the actual job request arrival over the next 50 sec interval. We observe that a very high degree of prediction accuracy is achievable by the LSTM networks because the prediction error is greatly reduced at each iteration by observing the actual job request arrival rates from the previous iteration. Fig. 3b shows the job request arrival rates used in the event-driven driven simulation by using OMNeT++.

APPENDIX B

OPTIMAL PROCESSOR SLICING IN CLOUDLETS

In this load balancing problem, each cloudlet uses *processor slicing* technique to handle multiple classes of incoming job requests with heterogeneous QoS target latency requirements, as shown in Fig. 4. We assume that each cloudlet has multiple processors and denote the total number of processors in i^{th} cloudlet by n_i . If the incoming job requests can be classified in to M classes, then the i^{th} cloudlet needs to create M slices with n_i^m processors in each slice such that $1 \leq n_i^m \leq n_i$. Firstly, each i^{th} cloudlet observes their individual incoming job request arrival rates for each class and finds the optimal values of n_i^m by solving the following optimization problem:

$$\begin{aligned} \mathcal{P}_i^m : \quad & \min_{(n_i^1, n_i^2, \dots, n_i^M)} \max_{m \in \mathcal{J}} \{t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m; n_i^m) - D_{Q_m}\} \\ & \text{subject to} \quad 1 \leq n_i^m \leq n_i, \sum_{m=1}^M n_i^m = n_i, \forall m \in \mathcal{J}. \end{aligned}$$

This optimization problem ensures an optimal processor slicing among all the job classes and allows soft processor slicing, i.e., n_i^m can take any real and ≥ 1 value. We formulate this problem such that a fair number of processors are allocated for all the job classes and avoid situations where the highest number of processors are allocated to a job class whose latency difference from D_{Q_m} can be extremely low but high for others. To solve the problem efficiently, we can reformulate the problem as follows:

$$\begin{aligned} \mathcal{R}_i^m : \quad & \min_{(n_i^1, n_i^2, \dots, n_i^M)} Z \\ & \text{subject to} \quad 1 \leq n_i^m \leq n_i, \sum_{m=1}^M n_i^m = n_i, \forall m \in \mathcal{J}, \\ & Z \geq (t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \lambda_i^m; n_i^m) - D_{Q_m}), \forall m \in \mathcal{J}. \end{aligned}$$

We can calculate the processing latency values as follows:

$$\mathbb{T}_i^m(\mu_i^m, \lambda_i^m; n_i^m) = \frac{1}{\mu_i^m} + \frac{\mathcal{E}_C(n_i^m, \lambda_i^m / \mu_i^m)}{n_i^m \mu_i^m - \lambda_i^m}, \quad (6)$$

(a) Inter-arrival times of the job requests from three different classes to Google clusters.

(b) Fitting job request inter-arrival times over 1 sec windows to exponential distribution.

Fig. 2: Inter-arrival times of the job requests from three different classes to Google clusters and their corresponding distribution fitting.

where, $\mathcal{E}_C(n_i^m, \lambda_i^m / \mu_i^m)$ is the *Erlang-C formula*, which can be written in an alternative form with $\rho_i^m = \lambda_i^m / (n_i^m \mu_i^m)$ for some advantage in the analysis as follows:

$$\begin{aligned} \mathcal{E}_C(n_i^m, \lambda_i^m / \mu_i^m) &= \frac{1}{1 + (1 - \rho_i^m) \left(\frac{n_i^m!}{(n_i^m \rho_i^m)^{n_i^m}} \right) \sum_{k=0}^{n_i^m-1} \frac{(n_i^m \rho_i^m)^k}{k!}} \\ &= \frac{1}{1 + \exp\left(\frac{\lambda_i^m}{\mu_i^m}\right) \left(n_i^m - \frac{\lambda_i^m}{\mu_i^m}\right) \left(\frac{\lambda_i^m}{\mu_i^m}\right)^{-n_i^m} \text{Gamma}\left(n_i^m, \frac{\lambda_i^m}{\mu_i^m}\right)}. \end{aligned} \quad (7)$$

Note that, we are using *incomplete Gamma function* in (7),

which is often used for approximate and exact representation of many Mathematical series [5]. The authors of [6] proved a conjecture that the average latency of $M/M/c$ queues are convex functions of the number of processors. They considered an expression like (6) and showed that it is a ratio of two expressions where the numerator is strictly decreasing and the denominator is strictly increasing with n_i^m . By further exploiting this property, they proved the convexity of this function. This implies that $\mathcal{R}_i^m$ is a continuous convex optimization problem and hence, it can be solved very efficiently with gradient projection algorithm.

(a) Prediction of job request arrival rates for two different job classes by LSTM algorithm with RMSE values 192 and 543, respectively.

(b) Job request arrival rates of two different classes to three cloudlets used in event-driven simulation with OMNeT++.

Fig. 3: Observed and forecasted job request arrival rates of two different classes to cloudlets.

APPENDIX C A DETAILED PROOF OF LEMMA 3.1

Proof: Recall that in the game Γ , the utility function of each i^{th} cloudlet $\mathcal{U}_i^N(\varphi_i, \varphi_{-i})$, $\forall i \in \mathcal{C}$ is defined as follows:

$$\begin{aligned} \max_{\varphi_i \in \Phi_i} \mathcal{U}_i^N(\varphi_i, \varphi_{-i}) = & \sum_{m=1}^M \Omega_{i,1}^m \frac{\lambda_i^m}{n_i^m \mu_i^m} \\ & + \sum_{m=1}^M \sum_{j=1, j \neq i}^N \Omega_{ji,2}^m \gamma_{ji} \frac{\varphi_{ji}^m \lambda_j^m}{n_i^m \mu_i^m} - \sum_{m=1}^M \sum_{j=1, j \neq i}^N \Omega_{ij,2}^m \gamma_{ij} \frac{\varphi_{ij}^m \lambda_i^m}{n_j^m \mu_j^m} \\ & - \sum_{m=1}^M \Omega_{i,3}^m \left[\left(1 - \sum_{j=1, j \neq i}^N \varphi_{ij}^m \right) \frac{\lambda_i^m}{n_i^m \mu_i^m} \max \{0, (t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \bar{\lambda}_i^m) - D_{Qm})\} \right. \\ & \left. + \sum_{j=1, j \neq i}^N \frac{\varphi_{ji}^m \lambda_j^m}{n_i^m \mu_i^m} \max \{0, (t_{uj}^m + \mathbb{T}_i^m(\mu_i^m, \bar{\lambda}_i^m) + t_{ji}^m - D_{Qm})\} \right] \end{aligned} \quad (8)$$

where $\bar{\lambda}_i^m(\varphi_i^m, \varphi_{-i}^m) = \{(1 - \sum_{j \neq i} \varphi_{ij}^m) \lambda_i^m + \sum_{j \neq i} \varphi_{ji}^m \lambda_j^m\}$. From this definition, we can observe that when i^{th} cloudlet is able to meet the QoS target latency D_{Qm} (under-loaded), or $(t_{uj}^m + \mathbb{T}_i^m(\mu_i^m, \bar{\lambda}_i^m) + t_{ji}^m - D_{Qm}) \leq 0$, then the utility function can be interpreted as an *affine function* as follows:

$$\begin{aligned} \mathcal{U}_i^N(\varphi_i, \varphi_{-i}) = & \sum_{m=1}^M \Omega_{i,1}^m \frac{\lambda_i^m}{n_i^m \mu_i^m} \\ & + \sum_{m=1}^M \sum_{j=1, j \neq i}^N \Omega_{ji,2}^m \gamma_{ji} \frac{\varphi_{ji}^m \lambda_j^m}{n_i^m \mu_i^m} - \sum_{m=1}^M \sum_{j=1, j \neq i}^N \Omega_{ij,2}^m \gamma_{ij} \frac{\varphi_{ij}^m \lambda_i^m}{n_j^m \mu_j^m}. \end{aligned} \quad (9)$$

Nonetheless, when i^{th} cloudlet is unable to meet D_{Qm} (overloaded), or $(t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \bar{\lambda}_i^m) - D_{Qm}) \geq 0$, then the utility function appears as a *non-linear function* as follows:

Fig. 4: A schematic diagram to illustrate the processing of job requests from multiple classes by processor slicing in a cloudlet.

$$\begin{aligned}
 \max_{\varphi_i \in \Phi_i} \mathcal{U}_i^N(\varphi_i, \varphi_{-i}) &= \sum_{m=1}^M \Omega_{i,1}^m \frac{\lambda_i^m}{n_i^m \mu_i^m} \\
 &+ \sum_{m=1}^M \sum_{j=1, j \neq i}^N \Omega_{ji,2}^m \gamma_{ji} \frac{\varphi_{ji}^m \lambda_j^m}{n_i^m \mu_i^m} - \sum_{m=1}^M \sum_{j=1, j \neq i}^N \Omega_{ij,2}^m \gamma_{ij} \frac{\varphi_{ij}^m \lambda_i^m}{n_j^m \mu_j^m} \\
 &- \sum_{m=1}^M \Omega_{i,3}^m \left[\left(1 - \sum_{j=1, j \neq i}^N \varphi_{ij}^m \right) \frac{\lambda_i^m}{n_i^m \mu_i^m} (t_{ui}^m + \mathbb{T}_i^m(\mu_i^m, \bar{\lambda}_i^m) - D_{Qm}) \right. \\
 &\left. + \sum_{j=1, j \neq i}^N \frac{\varphi_{ji}^m \lambda_j^m}{n_i^m \mu_i^m} (t_{uj}^m + \mathbb{T}_i^m(\mu_i^m, \bar{\lambda}_i^m) + t_{ji}^m - D_{Qm}) \right]. \quad (10)
 \end{aligned}$$

If (10) represents a concave function, then the sufficient condition is that its *Hessian matrix* should be a positive semi-definite matrix [7]. However, we find that both the diagonal and non-diagonal elements of this matrix are equal, i.e., $\left(\frac{\partial^2 \mathcal{U}_i^N}{\partial (\varphi_{ij}^m)^2} \right) = \left(\frac{\partial^2 \mathcal{U}_i^N}{\partial \varphi_{ij}^m \partial \varphi_{ik}^m} \right), i \neq j, k \in C, m \in \mathcal{J}$. Clearly, this implies that the Hessian of (10) is not a positive semi-definite matrix and we extend to evaluation of the *bordered Hessian matrix* [8]. Note that the r^{th} order bordered Hessian matrix of $\mathcal{U}_i^N(\varphi_i, \varphi_{-i}), \forall i \in C$, where $r = 1, 2, \dots, N$ is written as:

$$\mathcal{H}_B = \begin{bmatrix} 0 & \frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i1}^m} & \dots & \frac{\partial \mathcal{U}_i^N}{\partial \varphi_{ir}^m} \\ \frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i1}^m} & \frac{\partial^2 \mathcal{U}_i^N}{\partial (\varphi_{i1}^m)^2} & \dots & \frac{\partial^2 \mathcal{U}_i^N}{\partial \varphi_{i1}^m \partial \varphi_{ir}^m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{U}_i^N}{\partial \varphi_{ir}^m} & \frac{\partial^2 \mathcal{U}_i^N}{\partial \varphi_{ir}^m \partial \varphi_{i1}^m} & \dots & \frac{\partial^2 \mathcal{U}_i^N}{\partial (\varphi_{ir}^m)^2} \end{bmatrix}, \quad (11)$$

whose bordered elements are the first-order derivatives of $\mathcal{U}_i^N(\varphi_i, \varphi_{-i})$. Now, we observe that the determinant values of the bordered Hessian matrix (11), denoted by D_r , are negative

Fig. 5: A sample utility function of i^{th} cloudlet against job request offload fraction of neighboring j^{th} cloudlet φ_{ji}^m .

when r is odd and are positive when r is even, $\forall r = 1, 2, \dots, N$:

$$D_1 = 0 - \left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i1}^m} \right)^2 = - \left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i1}^m} \right)^2 \leq 0, \quad (12)$$

$$\begin{aligned}
 D_2 &= 0 - \left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i1}^m} \right) \left[\left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i1}^m} \right) \left(\frac{\partial^2 \mathcal{U}_i^N}{\partial (\varphi_{i2}^m)^2} \right) \right. \\
 &\quad \left. - \left(\frac{\partial^2 \mathcal{U}_i^N}{\partial \varphi_{i1}^m \partial \varphi_{i2}^m} \right) \left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i2}^m} \right) \right] \\
 &+ \left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i2}^m} \right) \left[\left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i1}^m} \right) \left(\frac{\partial^2 \mathcal{U}_i^N}{\partial \varphi_{i2}^m \partial \varphi_{i1}^m} \right) - \left(\frac{\partial^2 \mathcal{U}_i^N}{\partial (\varphi_{i1}^m)^2} \right) \left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i2}^m} \right) \right] \\
 &= - \left(\frac{\partial^2 \mathcal{U}_i^N}{\partial \varphi_{i1}^m \partial \varphi_{i2}^m} \right) \left[\left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i1}^m} \right) - \left(\frac{\partial \mathcal{U}_i^N}{\partial \varphi_{i2}^m} \right) \right]^2 \geq 0, \quad (13)
 \end{aligned}$$

and so on. Note that, the inequalities in (12)-(13) hold true, because we can observe that the second-order derivatives are all equal, i.e., $\left(\frac{\partial^2 \mathcal{U}_i^N}{\partial (\varphi_{i1}^m)^2} \right) = \left(\frac{\partial^2 \mathcal{U}_i^N}{\partial (\varphi_{i2}^m)^2} \right) = \left(\frac{\partial^2 \mathcal{U}_i^N}{\partial \varphi_{i1}^m \partial \varphi_{i2}^m} \right) = \left(\frac{\partial^2 \mathcal{U}_i^N}{\partial \varphi_{i2}^m \partial \varphi_{i1}^m} \right) \leq 0$, when the condition of stable operation $\{(1 - \sum_{j \neq i} \varphi_{ij}^m) \lambda_i^m + \sum_{j \neq i} \varphi_{ji}^m \lambda_j^m\} / (n_i^m \mu_i^m) < 1, \forall i, j \neq i \in C, \forall m \in \mathcal{J}$ holds. Therefore, we can conclude that (10) is a *quasi-concave function* of φ_i^m [8]. In general, we can conclude that the utility functions of each i^{th} cloudlet $\mathcal{U}_i^N(\varphi_i, \varphi_{-i}), \forall i \in C$ are affine functions of φ_i when they are under-loaded and quasi-concave functions of φ_i when they are overloaded. Hence proved. ■

It is interesting to note that in this load balancing game, each competing cloudlet is interested in *maximizing their individual utilities rather than strictly minimizing the average end-to-end latency* as most of the existing works. Hence, the cloudlets are always interested in receiving some job requests from neighboring cloudlets as long as the QoS latency requirement D_{Qm} is met and some extra incentive is gained. Fig. 5 shows that with a sufficient amount of job requests and a set of properly chosen parameters $\Omega_{i,1}^m$, $\Omega_{ij,2}^m$, and $\Omega_{i,3}^m$, the utility function i^{th} cloudlet monotonically increases as more job requests are offloaded by the neighboring j^{th} cloudlet until the total end-to-end latency is equal to the target QoS latency value D_Q . The maximum utility is achieved at the point where the total end-to-end latency is equal to D_{Qm} and the utility starts to

Fig. 6: Learning NE of the non-cooperative load balancing game with Algorithm 1 by two cloudlets with $n_1^m \mu_1^m = n_2^m \mu_2^m = 1000$ jobs/s, $\lambda_1^m = 970$ jobs/s, $\lambda_2^m = 800$ jobs/s, and $\Theta = 0.9$, $\sigma = 0.01$.

decrease beyond this point. Therefore, the overloaded cloudlets can only offload to its under-loaded neighboring cloudlets until their overall latency is equal to D_{Qm} . Thus, the fraction of incoming job requests offloaded by an overloaded cloudlet is controlled by the overloaded cloudlet as well as its under-loaded neighboring cloudlets.

Furthermore, note that in our game formulation, there is no explicit latency constraints on the federated cloudlets, even highly over-loaded cloudlets can participate in the game and can offload some of the job requests to the relatively under-loaded neighboring cloudlets. This makes their utility higher than the utility by not participating in the game. Nonetheless, under such conditions the game formulation in [9] that has explicit delay bound on participating cloudlets becomes infeasible and hence, a valid NE solution can not be obtained. As we compute the NE solution of this load balancing game, we can observe a very important property of the solution. The authors of [10] proved that the average latency expression of $M/M/c$ queuing systems is monotonically increasing and convex against load ρ . Intuitively, this implies that as the class- m job request arrival rate (λ_i^m) increases to i^{th} cloudlet, it needs to offload a higher fraction of job requests to its neighboring cloudlets for maintaining the stable condition of operation. Mathematically, if i^{th} cloudlet is overloaded such that $\hat{\lambda}_i^m > \lambda_i^m > 0$, then $\hat{\varphi}_{ij}^m > \varphi_{ij}^m > 0$.

APPENDIX D PROOF OF THEOREM 4.1

Proof: We considered that the strategy space Φ_i^m is a compact set in the load balancing game Γ and the stochastic reward values are normalized, i.e., $\mathcal{R}_i^m : \mathbb{R} \rightarrow [0, 1], \forall i \in \mathcal{C}, m \in \mathcal{J}$. The authors of [11] showed that with such conditions, a continuous action reinforcement learning automata-based PDF update rule,

$$f_i^{(n+1)}(\varphi_i) = \chi^{(n)} \left[f_i^{(n)}(\varphi_i) + \Theta^{(n)} (\mathcal{R}_i^{(n)} - \mathcal{R}_i^{(n-1)}) \exp \left\{ -\frac{1}{2} \left(\frac{\varphi_i - \varphi_i^{(n)}}{\sigma^{(n)}} \right)^2 \right\} \right], \quad (14)$$

can be guaranteed to converge to a local optimal NE, if the following necessary restrictions are imposed:

- (i) We choose a sufficiently small value of Θ^m such that each i^{th} cloudlet can match their expected strategy through iterations.

- (ii) We choose a value of σ^m such that the equilibrium point of $f_i(\varphi_i^m)$ has an upper bound $\frac{1}{\sigma^m \sqrt{2\pi}}$.

Thus, if the above restrictions are satisfied, then the continuous action reinforcement learning automata-based algorithm will converge to at least one of the existing NE of the underlying continuous-kernel game. ■

In Fig. 6, we show the convergence properties of the proposed reinforcement learning algorithm. We consider two neighboring cloudlets (just for illustration purpose), Cloudlet-1 and Cloudlet-2 with intermediate transmission latency $t_{ij}^m = 1$ msec, trying to meet a QoS requirement of $D_{Qm} = 10$ msec. We also consider that $n_1^m \mu_1^m = n_2^m \mu_2^m = 1000$ jobs/s and $\lambda_1^m = 970$ jobs/s, $\lambda_2^m = 800$ jobs/s, such that Cloudlet-1 is overloaded, but Cloudlet-2 is under-loaded. Therefore, to meet D_{Qm} , Cloudlet-1 needs to offload $0.083 \times \lambda_1^m$ job requests to Cloudlet-2, whereas Cloudlet-2 does not need to offload anything. From the PDFs of both the cloudlets also, we observe that the most preferable decision for Cloudlet-2 is not to offload and Cloudlet-1 prefers to offload around 8-9% of its total incoming job requests. As we choose the learning and spreading parameters as $\Theta = 0.9$ and $\sigma = 0.01$, respectively, we see that Algorithm 1 converges to the expected utility and reward values for both the cloudlets within a few hundred iterations. Note that, instead of searching over the whole strategy space, we considered only the most likely strategies that the cloudlets should possibly consider by using our understanding from the underlying load balancing game. Moreover, by using these techniques, Algorithm 1 can perform 100% accurately when all cloudlets are under-loaded without much exploration.

REFERENCES

- [1] L. Kleinrock, *Theory, Volume I, Queueing Systems*. Wiley-Interscience, 1975.
- [2] Google clusterdata 2011 traces. [Online]. Available: https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md
- [3] S. Chen, M. Ghorbani, Y. Wang, P. Bogdan, and M. Pedram, "Trace-Based Analysis and Prediction of Cloud Computing User Behavior Using the Fractal Modeling Technique," in *2014 IEEE International Congress on Big Data*, 2014, pp. 733-739.
- [4] C. Jiang, Y. Chen, Q. Wang, and K. J. R. Liu, "Data-Driven Stochastic Scheduling and Dynamic Auction in IaaS," in *2015 IEEE Global Communications Conference (GLOBECOM)*, Dec 2015, pp. 1-6.
- [5] P. Amore, "Asymptotic and exact series representations for the incomplete Gamma function," *Europhysics Letters (EPL)*, vol. 71, no. 1, pp. 1-7, jul 2005.
- [6] M. E. Dyer and L. G. Proll, "On the Validity of Marginal Analysis for Allocating Servers M/M/c Queues," *Management Science*, vol. 23, no. 9, p. 1019, 1977.
- [7] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.
- [8] R. E. Miller, *Optimization: Foundations and Applications*. John Wiley & Sons, Inc., 1999.
- [9] L. Chen, S. Zhou, and J. Xu, "Computation Peer Offloading for Energy-Constrained Mobile Edge Computing in Small-Cell Networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619-1632, Aug 2018.
- [10] H. L. Lee and M. A. Cohen, "A note on the convexity of performance measures of M/M/c queueing systems," *Journal of Applied Probability*, vol. 20, no. 4, p. 920-923, 1983.
- [11] A. Rodríguez, P. Vranx, R. Grau, and A. Nowé, "A Reinforcement Learning Approach to Coordinate Exploration with Limited Communication in Continuous Action Games," *The Knowledge Engineering Review*, vol. 31, no. 1, p. 77-95, 2016.