An adaptive rank-based approach for dynamic controller selection in Fog Computing

Marcus V. S. Costa[1,2], **Vitor B. Souza**[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais (IF SUDESTE MG)
Muriaé, MG – Brasil

[2]Departamento de Informática – Universidade Federal de Viçosa
Viçosa, MG – Brasil

marcus.costa@ifsudestemg.edu.br, vitorbs@dpi.ufv.br

Abstract. *The deployment of a dynamic and cooperative fog control plane, where controllers are selected on-demand among the most suitable underlying resources, has been recently proposed as a Control-as-a-Service (CaaS) model. Albeit it is expected that real-time applications shall benefit from this concept, mechanisms for QoS-aware controllers election is yet an open issue. In this work, we propose an adaptive rank-based controller selection strategy that is capable of tuning the weights employed for each characteristic of interest in order to cope with the environment dynamism. Results have shown an average controller exchange reduction of 37% when compared with a preliminary approach employing fixed weights in a dynamic scenario, as well as its efficiency in battery and memory usage by controllers.*

1. Introduction

In recent years, smart devices have significantly increased their ability to produce, process and store data, piquing the interest of governments, academia and industry for the development of innovative services, such as smart homes, smart cities, smart agriculture, green energy, e-health, disaster management, among others [Aazam et al. 2018]. Aligned with this evolution, the Internet of Things (IoT) brings a number of concepts, strategies and solutions related to devices located at the edge of the network. The number of deployed devices has reached impressive numbers in recent years and is expected to reach 29 billion connected devices by 2022 [Ericsson 2017]. Since these devices may have the ability to both perform remote sensing and operate over environments, data traffic in the network core is facing a huge increase due to the fact that data is traditionally stored and processed in large Data-Centers (DC) infrastructure offered by cloud service providers.

Transferring most of the computational effort required to support applications to the cloud may not be the most appropriate strategy in several cases. Applications requiring low service response time (SRT), such as real-time applications, may experience significant delays if the computational capacity is limited to the potential available at the core of the network. Indeed, the exclusive employment of cloud-based architectures may hinder the deployment of several IoT applications presenting strict QoS demands [Byers 2017].

Recent works have explored edge resource capabilities in order to deploy new approaches for task offloading. The rationale behind these approaches is that several applications may benefit from edge resources as an active part of data processing, reducing

communication latency due to the physical proximity among end-points. Fog computing is a novel edge computing paradigm that aims at extending the cloud to the edge of the network. Its main goal is to reduce communication delay whilst supporting system heterogeneity, mobility, and scalability [Bonomi et al. 2012]. Fog nodes process tasks collaboratively in a hierarchical architecture constituted by fog servers, clients, IoT devices, and users in close proximity, providing computational flexibility, storage, better communication, among others [Anawar et al. 2018].

In the fog control plane, edge controllers must discover underlying resources so that service requests may be mapped to the most suitable ones. However, the burden imposed on edge controllers is a challenging issue that is preventing fog providers to meet QoS requirements [Jiang et al. 2018]. Aligned to this, [Souza et al. 2017] proposed a distributed and collaborative control model, so-called Control-as-a-Service (CaaS), in order to dynamically deploy a QoS-aware control plane at the edge. Albeit the benefits coming from the deployment of on-demand controllers are shown in that work, authors do not propose a strategy for electing the best resources to play the controller role.

In [Costa et al. 2019], authors propose an approach for selecting, in real-time, the most suitable nodes to play as controllers in dynamic and heterogeneous fog domains. The rank-based approach weighs some key characteristics to measure a node's ability to become a controller while not affecting service performance and minimizing the overhead imposed by the need for controller exchanges. Although that work has presented satisfactory results regarding the convergence time and the number of controller exchanges, there is still room for significant improvement. Notice that fog dynamism and heterogeneity can generate an excessive number of controller exchanges on a short-term basis. Indeed, a node can have the highest rank value (RV) at one instant, hence, becoming controller, and have its RV decreased in a short time frame, being surpassed by another node, which becomes the new controller. On the other way around, if it subsequently suffers a new RV increase, it shall assume the controller role one more time. Excessive controller exchanges can increase overhead and degrade QoS. In addition, the original weight-based ranking scheme employs fixed weight for each node characteristic, not considering the intrinsic dynamism due to, for instance, the rate of moving nodes or with low battery. As an example, let us consider a scenario where most nodes are significantly constrained in terms of battery capacity. In such a case, the weight applied to this characteristic should be increased over others in order to prioritize the selection of candidates presenting higher battery levels.

In order to address this issue, this paper proposes an adaptive scheme where RV weights are tailored according to the fog scenario demands. Therefore, we propose an approach to select controllers on-demand in fog environments for CaaS provisioning, taking into account the environment characteristics to minimize the number of controller exchanges, decreasing overhead and better meeting QoS requirements. The main contributions of this work are:

- Extending the rank-based model proposed in [Costa et al. 2019] to dynamically tune the employed weights aiming at selecting the best resources to play the controller role according to environment constraints.
- Minimizing the frequency of controller exchanging in order to enable efficient CaaS provisioning whilst meeting QoS requirements in real-time applications.

This paper is organized as follows. Section 2 reviews resource selection proposals in distinct scenarios. Section 3 presents a enhanced rank-based approach for selecting controllers in dynamic and heterogeneous environments, such as in fog. In Section 4, we evaluate the proposed approach with experimental results. Section 5 concludes the paper and suggests avenues for future work.

2. Related work

A number of recent works have been devoted to solutions for the underlying resource selection problem. A widespread approach in the literature is to employ centralized controllers in distributed scenarios seeking to solve different objectives, for instance, optimizing the consumption of computational resources, reducing latency, network mapping, energy efficiency, just to name a few.

In Software Defined Networks (SDN) scenario, [Jiménez et al. 2015] propose a resource discovery protocol, so-called SDN-RDP, in order to allow controller nodes to discover the network topology by creating a control layer based on tree topology. An approach for controller selection in a wireless mesh SDN (wmSDN) architecture is proposed by [Salsano et al. 2014]. The proposal allows Wireless Mesh Routers (WMR) to select controllers through a priority list.

Works that relate controllers to switches in SDN are found to solve the QoS-guaranteed controller placement problem [Cheng et al. 2015] and minimize overall flow setup time in the network through a switch-controller mapping scheme [Sridharan et al. 2017]. The focus of both works is traditional SDN architectures, hence, switch and controller nodes are well defined.

In [Lee et al. 2017], the authors proposed a framework in which a fog node selects potential neighboring nodes in order to form a dynamic fog network for service offloading. Unfortunately, no strategy for selecting the first node is presented. A dynamic market game for fog environments was formulated by [Kim et al. 2018]. Authors evaluate economic aspects among end service-users (SU), edge resource owners (ERO) and ISP. The approach considers the ISP as a static controller leveraging dynamic edge resources for services allocation.

A cluster-based approach for resource discovery in Mobile Cloud Computing (MCC) is proposed in [Athwani and Vidyarthi 2015]. Battery power, signal strength, and mobility are parameters used to create a cluster-head (CH) selection function. Each CH manages resources in its cluster in order to select potential nodes for the service execution. Authors present no policy to update underlying cluster resources. [Arkian et al. 2014] also address the resource selection problem in vehicular cloud architecture. The cluster-based approach selects vehicles to perform the role of CH using fuzzy logic and reinforcement learning. Nodes' memory and the processing capacity are not evaluated for CH selection in both jobs, allowing to select CH with high processing capacity or low memory capacity, which can impair task offloading.

3. Enhanced model for controller selection

In this section, a rank-based strategy for selecting on-demand controllers in dynamic and heterogeneous distributed computing scenarios is presented. This strategy is an extension

of the one presented in [Costa et al. 2019]. The following subsection details each characteristic of interest (CI) considered for selecting potential controllers, then, it is presented how CIs are dynamically weighed in order to minimize controller exchanges and, finally, it is discussed on the process of selecting and exchanging controllers.

3.1. Identifying controller's requirements

In a distributed computing scenario, one of the main controller duties is to map service requests into the most suitable resources. The controller is not responsible for performing the service itself, but, in a highly dynamic and heterogeneous scenario, it must be able to keep an updated view of the underlying resource characteristics. Moreover, the selection of controllers should occur optimally so that it shall not impair the performance of the service. On the one hand, devices with powerful processors and larger memory are better suited for process offloading since service execution may be highly demanding. On the other hand, nodes endowed with low energy resources or presenting high displacement probabilities are more susceptible to unavailability and should not be employed as controllers, preventing QoS degrading.

Therefore, a strategy for selecting controllers has to consider the minimum requirements candidates should present to make them capable of processing and storing data referring to other nodes of the network, without compromising the execution of the service, prioritizing the control function. The key CIs considered in this work for selecting potential controllers are as follows.

- mobility: nodes with a higher probability of displacement are more likely to become unavailable, thus, they are less suitable to play the controller role;
- energy capacity: low battery power increases nodes volatility, reducing potential controllers availability;
- signal strength: low signal strength can increase packet loss, QoS degradation, and even connection disruption;
- processing and memory capacity: employing the most powerful devices as controllers may reduce the availability of powerful resources for service execution, increasing SRT.

Current edge devices can differ greatly in terms of memory, processing, power capacity, and mobility. Therefore, we propose a rank-based metric that weighs the heterogeneous CI mentioned above to select the best nodes to perform the controller role in a dynamic computing scenario.

3.2. RV calculation

Each node j has a set of CIs, denoted by $C_j = \{c_1, c_2, c_3, ..., c_n\}$, related to a set of weights $W = \{w_1, w_2, w_3, ..., w_n\}$, where n is the number of considered CIs, $0 \leq c_i \leq 100$ and $0 \leq w_i \leq 1$, where $1 \leq i \leq n$. Therefore, the RV of a candidate node j is given by

$$RV_j = \sum_{i=1}^{n} (c_i \times w_i) - p_j \tag{1}$$

where p_j is a punishment factor applied to a node j in order to minimize exchanges and is given by

$$p_j = \begin{cases} 0\,, & \text{if } j \text{ is the current controller} \\ max\{0, (\overline{RV_{diff}})^{-1} \times 100 - (R_c + R_n)\}\,, & \text{if } j \text{ has been a controller earlier} \\ 0\,, & \text{otherwise} \end{cases} \tag{2}$$

where $\overline{RV_{diff}}$ is the average difference between RV_j and $RV_{new\_controller}$ considering each time node j releases the controller role in favor of $new\_controller$, R_c is the number of consecutive rounds in which node j remained as controller within the last time it assumed that role, and R_n is the number of elapsed rounds since the last time node j released the controller role. According to (2), the current controller is not affected by the punishment factor. The strategy is to apply greater punishments to nodes that left the controller role with smaller $\overline{RV_{diff}}$ values or assumed as a controller only for a few rounds. These nodes are more likely to take over and lose the controller function many times in a few rounds and increase the number of exchanges.

As previously mentioned, this work considers processing, available memory, battery level, mobility, and signal strength as CIs for classifying nodes and selecting the best suited to perform the controller function. In the following lines, a comprehensive discussion on each CI is presented as well as how each one is quantified.

- Available memory (*M*) is the memory a candidate can share for executing controller duties. To play the controller role, a node must have enough memory to store information regarding resources shared by underlying nodes. The available memory value is classified into six levels. Therefore, *M* is given by

$$M = \begin{cases} 0, & \text{if } \mathrm{T}_{level1} \\ 20, & \text{if } \mathrm{T}_{level2} \\ 40, & \text{if } \mathrm{T}_{level3} \\ 60, & \text{if } \mathrm{T}_{level4} \\ 80, & \text{if } \mathrm{T}_{level5} \\ 100, & \text{if } \mathrm{T}_{level6} \end{cases} \tag{3}$$

- The mobility metric (*D*) sets higher scores to devices with reduced displacements. It is given by

$$D = 100 - F_d \tag{4}$$

  where F_d is a displacement factor of a node, which is a normalized variation of its coordinates – obtained by means of positioning techniques, such as GPS or triangulation – within a time interval $\Delta t = t_2 - t_1$. Therefore, F_d is given by

$$F_d = min\{100, \frac{100}{d_{max}}\sqrt{(x_{t2} - x_{t1})^2 + (y_{t2} - y_{t1})^2}\} \tag{5}$$

  where (x_{t1},y_{t1}) and (x_{t2},y_{t2}) are the location coordinates at the instants t_1 and t_2, respectively, while d_{max} is a predefined constraint for normalizing the computed distance according to the maximum displacement expected or allowed for a device playing the controller role within the interval Δt.
- Battery level (*B*) is computed by the percentage of total battery *b* currently available for a candidate – ranging from 1% to 100%. In addition, a controller candidate must have *b* greater than a predefined threshold b_{min}. Similarly, when a controller has *b* lower than b_{min}, a new controller must be selected to prevent the current one from running out of battery. Therefore, *B* is given by

$$B = \begin{cases} 100 \times b\,, & \text{if } b \geq b_{min} \\ 0\,, & \text{otherwise} \end{cases} \tag{6}$$

- To measure the signal strength (*S*) for each node, the Signal-to-Noise Ratio (SNR), normalized according to a predefined base value s_{max}, is used. Hence, *S* is given by

$$S = min\{100, \frac{100}{s_{max}} \times SNR\} \tag{7}$$

- The processing capacity (*P*) of each candidate is measured in MIPS. Similar to *M*, the value for processing performance is also classified into levels. However, for processing, eight distinct levels are proposed where the maximum value (*P*=100) is attributed to P_{level4} whilst the attributed value is reduced as the processing capacity is either increased or decreased. Hence, *P* is given by

$$P = \begin{cases} 0, & \text{if } \mathrm{P}_{level1} \\ 50, & \text{if } \mathrm{P}_{level2} \\ 75, & \text{if } \mathrm{P}_{level3} \\ 100, & \text{if } \mathrm{P}_{level4} \\ 90, & \text{if } \mathrm{P}_{level5} \\ 60, & \text{if } \mathrm{P}_{level6} \\ 20, & \text{if } \mathrm{P}_{level7} \\ 5, & \text{if } \mathrm{P}_{level8} \end{cases} \tag{8}$$

On the one hand, P_{level2} is considered the minimum processing capacity required for a candidate to be used as a controller. On the other hand, P_{level4} is the desired processing capacity for the proper execution of controller tasks without loss of QoS due to processing overhead. Our goal is to allocate resources that comply with the minimum processing requirements for controllers while preserving the most powerful resources for the service execution.

Therefore, the complete equation for obtaining the RV of each node *j* is given by

$$RV_j = ((D_j \times w_D) + (M_j \times w_M) + (B_j \times w_B) + (S_j \times w_S) + (P_j \times w_P)) - p_j \tag{9}$$

where w_D, w_M, w_B, w_S and w_P are the respective weights applied to D_j, M_j, B_j, S_j and P_j.

3.3. Computing RV weights dynamically

As described in the previous subsections, calculating the RV consists of obtaining and classifying nodes' CIs followed by each weight assignment. This is performed after every time period (hereinafter referred to as rounds) when underlying nodes send their CIs to the current controller. In this work, we propose to define weights dynamically, after each round, according to the scenario's demands. This is a challenging task because, in very dynamic environments, CIs can switch quickly. Furthermore, acknowledged the heterogeneity of underlying devices, an improper strategy for weighing CIs shall, for sure, lead to under-utilization of available resources due to the selection of bad candidates to play as controllers, violating QoS requirements.

Figure 1 shows an analysis of some CIs variation for five random nodes within two hundred simulation rounds. The observed nodes have high displacement probability, impacting on a large variation of SNR and the employed mobility metric (see (4) and (5)), as shown respectively in Figure 1(c) and Figure 1(d). On the other hand, albeit available

memory (see Figure 1(b)) has had a high variation in some nodes, its contribution to large variations in RV tends to be considerably smaller. Therefore, to minimize unnecessary exchanges due to RV fluctuation, the weights assigned for D and S must be lower. Nevertheless, these CIs cannot be neglected as they are important to assess a node availability at a given time, hence, the respective weights should not be underestimated.

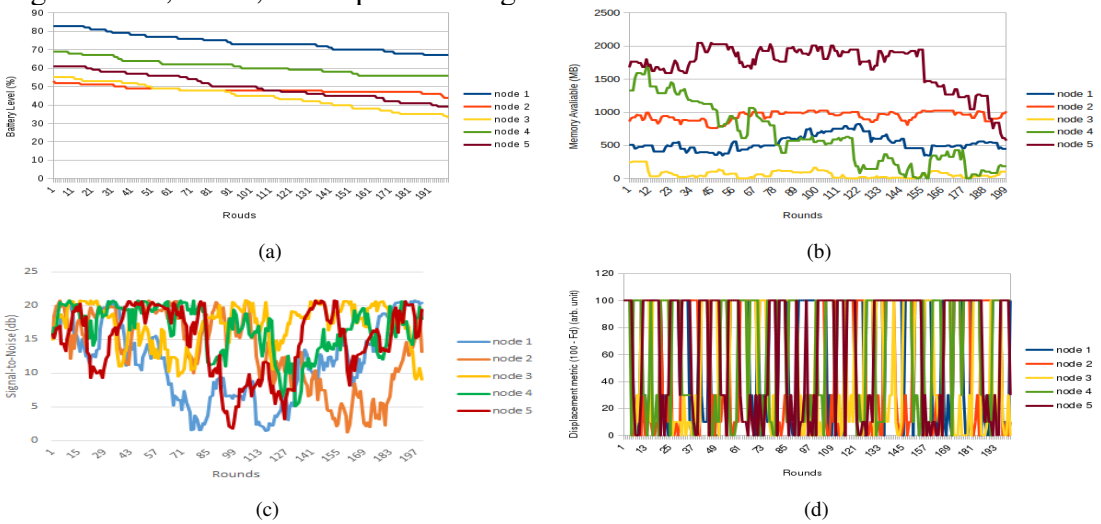

Figure 1. Variation of CIs in arbitrary nodes: (a) Battery level, (b) Available memory, (c) SNR and (d) Mobility.

In the deployed strategy, the weight applied to the node's mobility (w_D) is scenario-dependent. Hence, the higher the number of nodes with low signal strength, the higher the weight attributed to this CI. The rationale behind this approach is to prioritize nodes with low mobility since there is a higher chance of increasing packet loss due to mobility. Moreover, this strategy reflects the need for prioritizing controllers offering both high availability and, as far as possible, large available memory. The weights dynamically applied to RV computation are given by

$$\begin{cases} w_D \text{ (mobility)} & = & 0.10 + (R_{smin} \times 0.05) \\ w_B \text{ (battery level)} & = & 0.35 + (R_{bmin} \times 0.05) \\ w_M \text{ (memory available)} & = & 0.35 - w_D \\ w_S \text{ (signal strength)} & = & 0.50 - w_B \\ w_P \text{ (processing capacity)} & = & 1 - (w_D + w_B + w_M + w_S) \end{cases} \quad (10)$$

where R_{smin} is the ratio of nodes presenting S less than a predefined $low\_signal\_strength$ threshold and R_{bmin} is the ratio of nodes presenting B less than a predefined $low\_battery\_level$ threshold. Notice that a tradeoff comes to light according to the number of nodes above R_{smin} and R_{bmin} thresholds. In fact, the weight applied for mobility and available memory is inversely proportional. The same occurs between the battery and signal strength. In this particular case, a balance is proposed: the battery may increase or maintain its priority over the node's indicators of displacement (signal strength and mobility) proportionally to the number of nodes below the threshold R_{bmin}.

3.4. Controllers selection

This subsection details the decentralized controller selection process for CaaS provisioning as well as the controller exchanging process. Initially, each node calculates its own

RV. Since, originally, candidates do not know each other, it is not possible to tailor weight values according to the current scenario prior to the first controller selection. Therefore, all nodes assume predefined weight values to compute their RV for the first time. Every node presenting one or more CIs equal to zero gives up the selection process. Additionally, since mobility computing takes a time Δt, it is not considered in the first controller election, preventing latency increase. Hence, the mobility weight is initially set to 0 (zero).

Each candidate node broadcasts its calculated RV and waits for the reception of other candidates RVs. Due to the large variation expected on the number of devices, each controller candidate sets a timeout interval in which it listens to other candidates' messages. Upon receiving a RV from another node, the candidate compares it with its own RV. If it is greater than or equal to the received one, it resets the timeout timer and continues to wait for messages from other candidates. Otherwise, it gives up being a controller by canceling the scheduled timeout event.

The candidate with the highest RV shall be the only node persisting in the selection after message exchanges. As soon as the timeout event is triggered, the node broadcasts its controller status. With this approach, a candidate must reset its timeout at most *n-1* times before becoming a controller, where *n* is the number of candidates. In the hypothesis of candidates presenting the same RV, the contention is resolved through the highest battery level. Involved nodes exchange extra messages containing the battery level and an extra 2 bytes random number to be used in case of equal battery levels. In this case, the node with the largest random number wins the contention. If two or more nodes announce themselves as controllers at the end of the selection process, the same tie-breaking process is employed.

After the elected controller broadcasts its status, the resource discovery phase starts. Within one round, each non-controller node unicasts a message to the controller informing its characteristics, including the CIs used for the controller selection and its punishment factor (calculated by the nodes). Thus, the controller builds its local database so it can map received requests into the most suitable resources according to the service requirements. It is worth mentioning that extra characteristics may be included apart from the characteristics used for controller selection, but, since they are service-specific, they are out of the scope of this work.

In addition, non-controller nodes send their CI and punishment factor at each round enabling the controller to keep an up-to-date database. At the end of each round, the controller computes both the amount of nodes with a low battery level and the amount of nodes with signal strength bellow the defined threshold and determines the weights for each CI. Then, the weights are employed to calculate the RV of each node, including its own. If one node has a RV higher than the controller's one, the controller unicasts a message informing it shall be the next controller. After acknowledging the message, it announces itself as the new controller. The message sent by the previous controller informs which weights were used in the last round so that the new controller updates its weight values.

In order to prevent $(\overline{RV_{diff}})^{-1}$ from assuming high values, making it impossible for a node to be a controller candidate for hundreds of rounds, controller exchanging

takes place only if the node's RV is higher than the controller's RV in at least one unit. The controller exchange is detailed by Algorithm 1. If a tie occurs when selecting the new controller, the same criteria for solving a contention is employed. However, if nodes have the same battery level, the current controller randomly chooses its successor.

Algorithm 1 Controller Exchange

```
1: l: controller node
2: N: set of non-controller nodes
3: t: predetermined round time
4: h: arbitrary node
5: procedure CONTROLLEREXCHANGE( )
6:     while True do
7:         for each node j in N do
8:             l collects j's CI and j's pf within a time t          ▷ performs concurrently
9:         l determines the weights that will be applied to the RV
10:        l calculates all nodes' RV including its own
11:        h ← node with highest RV in l's database
12:        if h's RV > l's RV then                                  ▷ h can be a new controller
13:            l informs h that it is the new controller and the weights used in the last round
14:            break
15:    h broadcasts its status to N
16: end
```

4. Performance analysis and comparative study

This section presents the experiments deployed for evaluating the proposed strategy. The experiments assess the efficiency in reducing controller exchanges and aspects involving QoS requirements. For instance, controllers are evaluated in terms of both availability and capacity to store underlying resources information. In addition, the controller capability to keep an updated view of those resources CI is further evaluated. For each experiment, we compare, through obtained results, the proposed approach with a tailored version of a resource selection strategy available in the literature.

4.1. Comparative resource selection strategy

For the sake of comparison, a CH selection strategy devoted to MCC scenarios, described by [Athwani and Vidyarthi 2015], was tailored in order to fulfill the scenario considered in this work. In the presented cluster-based approach, each selected CH is responsible for acquiring and sharing knowledge about the resources available in the respective cluster. For comparison, the proposed scheme is adapted from ad-hoc to an infra-structured communication model. For instance, in the original model, a relative mobility function is computed through the variation of the signal strength between each pair of neighboring nodes. In the infra-structured communication model, the mobility is calculated individually by each node by means of the variation on the strength of beacons received from the access point (AP). Notice that, since the interval between beacon messages is too short, T_s is employed as the minimum time between two beacons considered for computing signal variation. The adapted version is hereinafter referred to as a cluster-based model.

Nodes are elected by a cluster function given by

$$Clus_{func} = w * Mob + (1 - w) * (B_{power})^{-1} \tag{11}$$

where Mob is given by the node mobility function, B_{power} is the battery power of a node, w is a weight factor, subject to $0 \leq w \leq 1$. Each node broadcasts its $Clus_{func}$ value and the one with lower value is selected as CH.

Since the focus of this work is not on cluster formation, only one single cluster is considered, with one single CH at a time. Another important difference is that the former work does not consider the need for reelection when nodes' parameters change over time. Rather than that, a new CH takes place only after a node failure or if a node with a better $Clus_{func}$ value gets in the cluster. As a result, it reduces the amount of controller exchanges, however, QoS degradation is expected since, before failure, a controller may suffer from low throughput and memory overload for some period, depending on the cause of failure. Consequently, the controller may relinquish service requests.

4.2. Scenario Description

The conducted experiments emulate a fog environment with dynamic and heterogeneous devices. Each edge node is emulated by a virtual machine whilst the IEEE 802.11b/g connection among them is emulated by means of NetEm [Foundation 2018].

In this work, the same criteria adopted in [Costa et al. 2019] is used to classify the nodes regarding their displacement profile, as shown in table 1. In this classification, type 1 nodes are more likely to suffer large variations in RV due to mobility and varying signal strength.

Table 1. Nodes Classification

Node Type	Displacement Probability	Examples
Type 1	High	Vehicles, drones
Type 2	Moderate	Smartphones, embedded devices
Type 3	Low	PCs, micro-servers

Regarding the heterogeneity of nodes characteristics, each node is endowed with processing capacity ranging from 1500 to 35000 MIPS and available memory ranging from 150MB to 3GB. Available memory, signal strength, mobility and battery level are features that can vary throughout simulation time. The memory variation is simulated through a probability density function (PDF) in inverse gamma distribution with shape parameter $\alpha = 0.5$ and scale parameter $\beta = 10$. A random mobility model is used while SNR is calculated according to the current location of the node. For battery consumption, a linear variation is employed. Table 2 shows the main parameters employed. It is worth mentioning that the weights shown in the table are used for setting up the emulation start and shall vary throughout the experiment time. The initial weights are inferred from [Costa et al. 2019]. Table 3 shows the adopted classification for memory and processing capabilities of nodes.

4.3. Results

In this section, the proposed approach is assessed in three different aspects: the number of controller exchanges, the accuracy of controller knowledge regarding dynamic node resources, and the availability of resources on employed controllers. Three distinct scenarios are considered for the proposed model.

- Scenario 1 is constituted by nodes with high displacement probability presenting a distribution of 50% type 1 nodes and 50% type 2 nodes.
- Scenario 2 is constituted by 50% of type 3 nodes, 25% of type 1 nodes, and of 25% type 2 nodes.
- Scenario 3 is fully constituted by devices with a low probability of movement, i.e., 100% of type 3 nodes.

Table 2. Experiment parameters

Number of nodes	100
Experiment time	900 rounds
w_D (weight for mobility)	0.2
w_B (weight for battery level)	0.25
w_M (weight for memory available)	0.25
w_S (weight for signal strength)	0.1
w_P (weight for processing capacity)	0.2
b_{min}	10%
$\Delta t = T_s$	5 rounds
d_{max}	100m
s_{max}	20db
$low\_signal\_strength$ (relative to S metric)	35
$low\_battery\_level$	30%
w (weight factor for cluster-based model)	0.5

Table 3. Available memory (M) and processing capacity (P) levels

Memory		Processing	
M Level	Memory (MB)	P level	MIPS
T_{level1}	less than 128	P_{level1}	less than 4375
T_{level2}	from 128 to 256	P_{level2}	from 4375 to 8750
T_{level3}	from 257 to 512	P_{level3}	from 8751 to 13125
T_{level4}	from 513 to 768	P_{level4}	from 13126 to 17500
T_{level5}	from 769 to 1024	P_{level5}	from 17501 to 21875
T_{level6}	more than 1024	P_{level6}	from 21876 to 26250
		P_{level7}	from 26251 to 30625
		P_{level8}	more than 30625

In the cluster-based model emulation, all nodes are considered to have a high probability of movement since the work in which it is based is focused on scenarios with high mobility. Therefore, the nodes' configuration is similar to Scenario 1, i.e., 50% are type 1 nodes while 50% are type 2 nodes.

In the first experiment, the amount of controller exchanges is evaluated. For comparison purposes, both versions of the proposed model are considered. The original version ([Costa et al. 2019]), hereinafter referred to as a former model, makes use of fixed weights and no punishment factor, whilst the one proposed in this work implements the punishment factor and adaptive weights. The amount of controller exchanges in the cluster-based model, the former model, and the proposed adaptive model is shown in Figure 2. For the former and adaptive models, three distinct scenarios are considered.

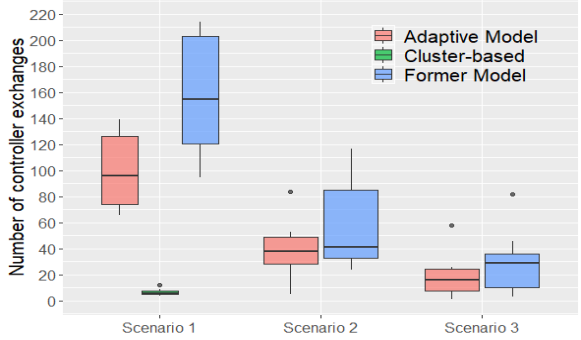

Figure 2. Controller exchanges

As expected, fewer changes are observed in the cluster-based model since changes only occur when the connection fails or battery discharges. The adaptive weights strategy for computing the RV, in collaboration with the punishment factor, led to a decrease of approximately 37% on the amount of controller exchanges in scenario 1, which is more conducive to exchanges, compared to the use of fixed weights. Some examples of the variation of the employed weight within the experiment time are shown by Figure 3.

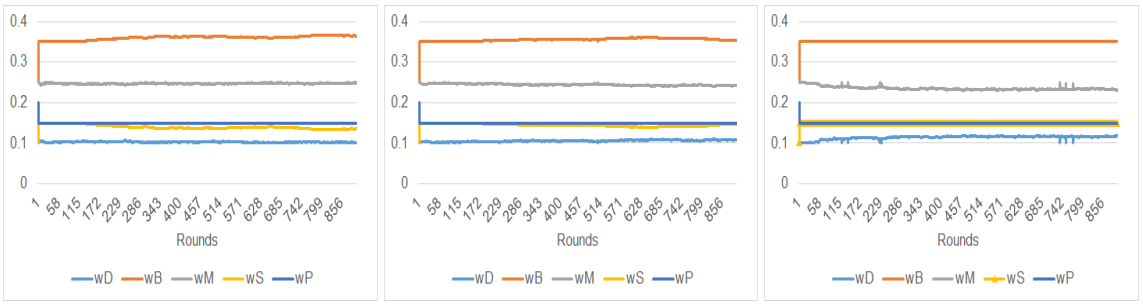

Figure 3. Weight variation. (a) Scenario 1, (b) Scenario 2 and (c) Scenario 3

A mandatory requirement for QoS-aware controllers is to keep up-to-date information regarding highly dynamic resources so that the most suitable ones can be employed for supporting service requests. In the second experiment, the adaptive model is compared with the cluster-based model on this behalf. Distinct characteristics collected by controllers from one non-controller node are shown in Figure 4, where a blue line shows the actual value for each characteristic. Within 500 rounds, 70 samples of each CI were randomly collected for both cluster-based and adaptive approaches. The proposed adaptive approach presents better accuracy regarding the variations of node characteristics than the comparative method. Indeed, the cluster-based approach expects edge nodes to send messages about their available resources only when a new CH takes place.

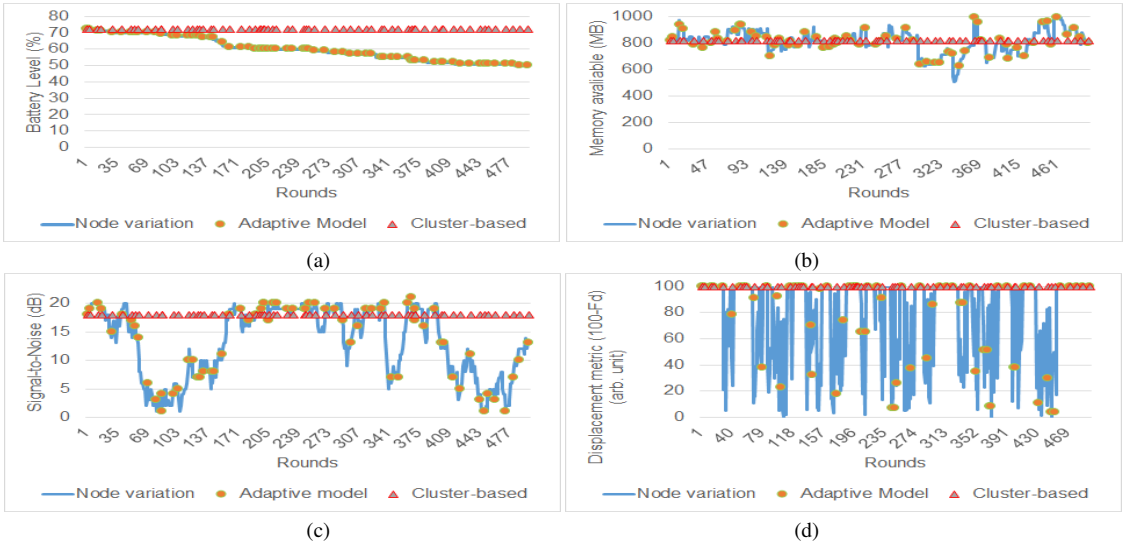

Figure 4. Accuracy of controller knowledge of a node's dynamic resources (a) Battery level, (b) Available memory,(c) Signal-to-Noise and (d) Mobility

A critical requirement is that resources selected to play the controller role do not adversely affect service performance. In the third experiment, characteristics of all de-

ployed controllers, collected at each round, are compared. Figure 5 shows battery and memory availability in controllers. As can be seen from the results, the proposed approach is more efficient in selecting potential controllers for all scenarios evaluated than the comparative method. Selecting controllers with higher battery levels enables them to provide CaaS longer. In addition, controllers with more available memory are more capable of managing large databases of edge resources.

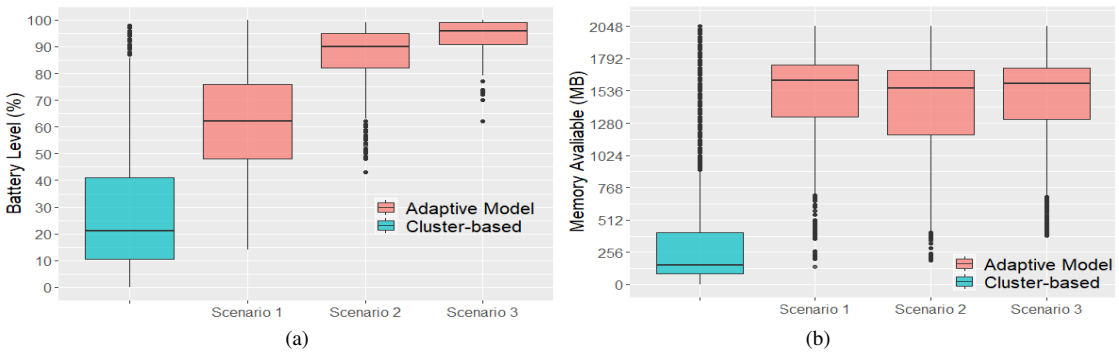

Figure 5. (a) Battery level and (b) Memory availability on controllers

5. Conclusion

This work proposed an adaptive weighting strategy for rank-based on-demand controller selection aiming at CaaS provisioning in dynamic and heterogeneous scenarios, such as fog. The proposed model makes use of dynamic weights for the characteristics considered for selection in order to elect QoS-aware controllers whilst minimizing the number of unnecessary controller exchanges. In the performed experiments, this strategy has proved its efficiency when compared with a static weight approach. When compared to a cluster-based model present in the literature, albeit the amount of controller exchanges has shown to be higher, the model proposed by this work has overcome the cluster-based approach both in terms of accuracy of underlying resources information, and efficiency on selecting resources for playing the controller role, offering QoS-aware service allocation. As future work, we aim at addressing the increasing overhead inherent to the need for keeping an updated view of the available resources. We also aim at providing resilience to cope with unexpected controller failures.

Acknowledgment

This work is supported by FAPEMIG, CNPq, and CAPES.

References

Aazam, M., Zeadally, S., and Harras, K. A. (2018). Offloading in fog computing for iot: Review, enabling technologies, and research opportunities. *Future Generation Computer Systems*, 87:278 – 289.

Anawar, M. R., Wang, S., Zia, M. A., Jadoon, A. K., Akram, U., and Raza, S. (2018). Fog computing: An overview of big iot data analytics. *Wireless Communications and Mobile Computing*, 2018:7157192:1–7157192:22.

Arkian, H. R., Atani, R. E., and Pourkhalili, A. (2014). A cluster-based vehicular cloud architecture with learning-based resource management. In *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, pages 162–167.

Athwani, P. and Vidyarthi, D. P. (2015). Resource discovery in mobile cloud computing: A clustering based approach. In *2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON)*, pages 1–6.

Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. (2012). Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA. ACM.

Byers, C. C. (2017). Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks. *IEEE Communications Magazine*, 55(8):14–20.

Cheng, T. Y., Wang, M., and Jia, X. (2015). QoS-guaranteed controller placement in SDN. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6.

Costa, M. V. S., Souza, V. B., and Júnior, S. S. A. (2019). Dynamic control-as-a-service provisioning in fog computing. In *2019 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6.

Ericsson (2017). Ericsson mobility report. `https://www.ericsson.com/assets/local/mobility-report/documents/2017/ericsson-mobility-report-june-2017.pdf`. Accessed: 2019-10-08.

Foundation, L. (2018). Linux foundation wiki: netem. `https://wiki.linuxfoundation.org/networking/netem`. Accessed: 2019-10-14.

Jiang, Y., Huang, Z., and Tsang, D. H. K. (2018). Challenges and solutions in fog computing orchestration. *IEEE Network*, 32(3):122–129.

Jiménez, Y., Cervelló-Pastor, C., and García, A. (2015). Dynamic resource discovery protocol for software defined networks. *IEEE Communications Letters*, 19(5):743–746.

Kim, D., Lee, H., Song, H., Choi, N., and Yi, Y. (2018). On the economics of fog computing: Inter-play among infrastructure and service providers, users, and edge resource owners. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6.

Lee, G., Saad, W., and Bennis, M. (2017). An online secretary framework for fog network formation with minimal latency. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6.

Salsano, S., Siracusano, G., Detti, A., Pisa, C., Ventre, P. L., and Blefari-Melazzi, N. (2014). Controller selection in a wireless mesh SDN under network partitioning and merging scenarios. *CoRR*, abs/1406.2470.

Souza, V. B., Gomez, A., Masip-Bruin, X., Marin-Tordera, E., and Garcia, J. (2017). Towards a fog-to-cloud control topology for qos-aware end-to-end communication. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pages 1–5.

Sridharan, V., Gurusamy, M., and Truong-Huu, T. (2017). On multiple controller mapping in software defined networks with resilience constraints. *IEEE Communications Letters*, 21(8):1763–1766.