Enhancing Service Management Systems with Machine Learning in Fog-to-Cloud Networks

Jasenka Dizdarević(✉), Francisco Carpio, Mounir Bensalem, and Admela Jukan

Technische Universität Braunschweig, Braunschweig, Germany
{j.dizdarevic,f.carpio,m.bensalem,a.jukan}@tu-bs.de

Abstract. With the fog-to-cloud hybrid computing systems emerging as a promising networking architecture, particularly interesting for IoT scenarios, there is an increasing interest in exploring and developing new technologies and solutions to achieve high performances of these systems. One of these solutions includes machine learning algorithms implementation. Even without defined and standardized way of using machine learning in fog-to-cloud systems, it is obvious that machine learning capabilities of autonomous decision making would enrich both fog computing and cloud computing network nodes. In this paper, we propose a service management system specially designed to work in fog-to-cloud architectures, followed with a proposal on how to implement it with different machine learning solutions. We first show the global overview of service management system functionality with the current specific design for each of its integral components and, finally, we show the first results obtained with machine learning algorithm for its component in charge of traffic prediction.

Keywords: Machine learning · Fog-to-Cloud · Service management

1 Introduction

In the recent years, integration of the fog and cloud computing into fog-to-cloud hybrid computing systems has became an important research subject, especially regarding their presence in Internet of Things (IoT) scenarios. Both cloud and fog computing satisfy different system requirements, complementing each other. In these integrated solutions, cloud servers are used for analyzing and processing large amounts of data that require high computing power and where service execution is not time sensitive. Fog nodes include less powerful devices, but also with computing power and data storage capabilities, which allows them to process data from multiple sensors while minimizing latency and reducing the amount of data which needs to be transported to the cloud. Some of the efforts devoted to the development of an integrated fog to cloud (F2C) system include the OpenFog Consortium [1] and the mF2C H2020 EU project [2].

© Springer Nature Switzerland AG 2019
G. Mencagli et al. (Eds.): Euro-Par 2018 Workshops, LNCS 11339, pp. 287–298, 2019.
https://doi.org/10.1007/978-3-030-10549-5_23

In order to ensure high performance of F2C solutions it is necessary to explore new emerging technological solutions, such as Machine Learning (ML). At the moment there is no determined definition of how machine learning should be used in a F2C system, but it is evident that both fog and cloud enriched with ML capabilities would improve the system performances with the ability to make decisions and take actions individually based on algorithmic sensing of patterns in locally captured sensor data in the case of fog nodes, and centrally captured data in the case of the cloud nodes. In this paper, we propose ML implementation as a mean of improving a specific area of F2C - service management. Since the architecture of a F2C system is still a relatively new research field, without strictly defined standards and guidelines, in order to observe ML behavior for this purpose, we also propose an architectural component of F2C that would be in charge of service related functionalities.

With most of the research efforts in fog computing and fog-to-cloud based systems being more focused on the integrated architecture and communication aspects of these new systems, and not just on the particular problem of managing services, we relied on previous work done in this area in order to propose a design of our service management component. A comprehensive survey on service management and handling Web services and distributed services was conducted in [3], offering an overview of well known *service-oriented architectures* concepts. Some of the available service management solutions are cloud oriented such as [4,5] or they present completely novel approaches as the one proposed in [6], where authors developed a concept of managing services that simplifies service operations by sharing different tasks and functionalities of a global service among multiple distributed agents. Also, some papers focus on the service management in IoT solutions, so for example in [7] authors propose *Management Server Service* as a part of their IoT system architectural design for handling service related tasks.

In this paper, we propose possible ways of using ML in a specific part of fog-to-cloud computing system - components that are in charge of managing service related functionalities. For this purpose, we propose a Service Management System (SMS) integral unit, composed of multiple components, each of them representing a different functionality. In the following sections we analyze possible ML application areas in these components.

The rest of this paper is organized in the following way. Section 2 introduces the Service Management System for fog-to-cloud based systems. Section 3 covers possible Service Management System components where different machine learning algorithms can be used and show the preliminary results with Sect. 4 concluding the paper.

2 Service Management System

As the main F2C architecture component in charge of service related functionalities, we propose a Service Management System (SMS), shown in Fig. 1, as an integral component that can be deployed on all nodes/devices with computing

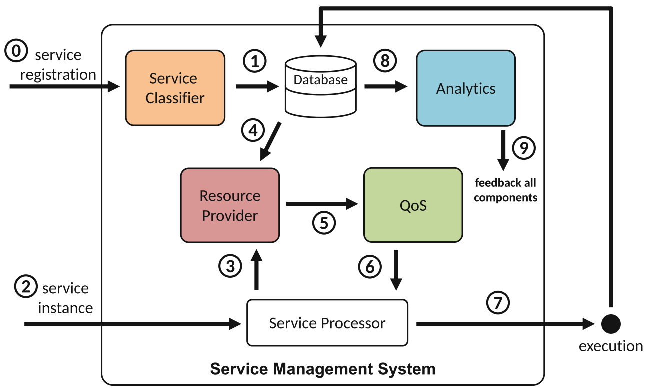


Fig. 1. Service Management System architecture

capacities, both in the cloud and the fog. Different SMS tasks include classifying different types of services, executing different phases of the service lifecycle, deciding in which F2C system node will the execution take place, gathering and measuring network data from different nodes, and ensuring quality of the service.

Figure 1 shows the components that comprise the Service Management System: Service Classifier, Service Processor, Resource Provider, QoS, Analytics and a Database. When a new service registers to the system, regardless of the node, it will first be passed on to the Service Classifier component where different services will be classified based on different requirements they have, and this information will be saved in a database. Once the service is registered, the system will be ready for receiving new instances of the service for executions. When one of these service instances arrive, the Service Processor component of SMS, which controls the service lifecycle, have to be previously registered. The Service Processor will then communicate with the Resource Provider, the component that decides where a service instance will be executed. For this decision, it first has to read the information from the database on the availability of the nodes, where he can obtain the information, whether it is possible to use multiple nodes, as well as the information from the Analytics on traffic prediction. Nodes represent different devices with different levels of computing and processing capabilities, allocated in different abstract layers of F2C system. To decide whether it is possible to use these nodes, it will contact the QoS component. Based on the recommended nodes from the Resource Provider and the possibility of using them, the Service Processor will get the information where the service instance should be executed and deploy it accordingly. The results of the execution are saved in the database, so the Analytics component can use them to update the Resource Provider and the QoS.

In the following section we will propose how some of these component's functionalities can be improved with machine learning solutions, which should result with the improvement of the entire F2C system performances.

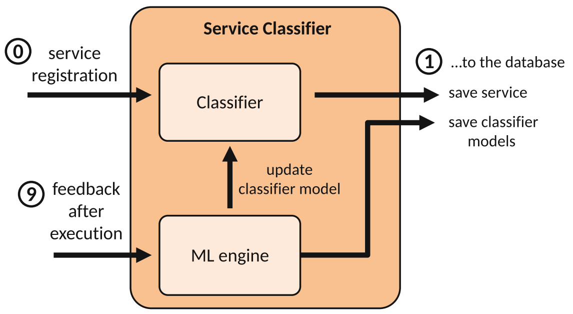


Fig. 2. Service Classifier component

3 Components Implementing Machine Learning Algorithms in Service Management System

In the Service Management System, we propose different methods of implementing machine learning in the components where traditional heuristic algorithms are not enough for taking complex decisions. These components include Service Classifier, Resource Provider, QoS and Analytics. The numerical results are shown for the Analytics and its implementation of ML for traffic prediction, while for the other components we propose the current design, with numerical results being the next step to prove the validity of the design.

3.1 Service Classifier

In [9] a service classification was proposed based on user defined requirements. The goal was service differentiation based on their requirements to be able to allocate resources. The service classification method proposed here was grouping the services into classes according to information defined by the user. However, what was not taken into consideration, and in most service classification modules, is that service requirements can be different from one execution to another which makes the classification process dynamic and non trivial. As a result, we need to learn from a previous execution of the service in the network to achieve an accurate classification. The Service Classifier, shown in Fig. 2, is responsible for the categorization of new services registered into the F2C system based on the information specified by the user about the service requirements. Some requirements necessary for the service to be successfully executed are unknown to the user, such as the network load, network topology, resource load, etc. Thus, in the beginning the Classifier needs to execute the service in the network and then extract some information to enhance the classification process. Afterwards, service categories are stored in the database.

After the test executions, the Analytics component, which collects information about services and node performance, will contact Service Classifier in order to feedback the results from the execution which will update the ML engine for

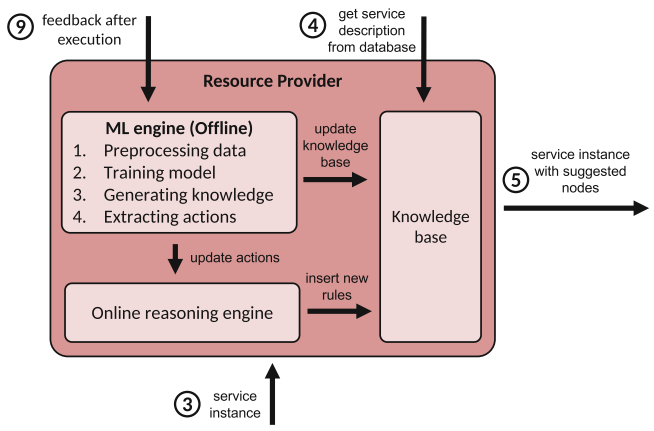


Fig. 3. Resource Provider component

the future classifications. The ML engine, in response, preprocesses the data and extracts the features that will then train and tune a model offline. The model obtained is used to update the Classifier which works online with new incoming services. Moreover, it is also used to update the categories that are already stored in the database. For the next step the presented component design should be implemented with the choice of an adequate ML algorithm in order to test the effectiveness and performances.

3.2 Resource Provider

The service instances that have already been registered, as described previously, upon arriving for execution to the Service Processor component of SMS, initialize the communication between the Service Processor and the Resource Provider, the component that decides where the execution will take place.

Figure 3, shows the global picture of the Resource Provider component, which includes three main parts: a knowledge base (KB), an online reasoning engine and a ML engine working offline. So, when the Service Processor receives a service instance, it requests for a node recommendation from the Resource Provider. The Resource Provider will obtain this information about the particular service from the database, generate node suggestions and send them to the QoS component to decide where to execute this service instance. When a new service instance is requested, the online reasoning engine generates recommendations based on predefined rules and then stores the recommendations for each service instance in the KB. If the information on the instance already exists in the KB, the Resource Provider will directly send the suggestions to the QoS. After the execution, the system collects statistics about the network: feedback about the suggested nodes, traffic prediction, holding time, etc.

The gathered data is used to feed the ML engine working offline which is responsible for recognizing data patterns to improve the reasoning engine and

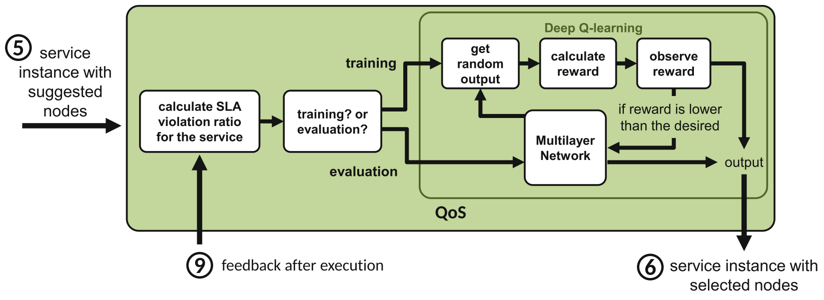

5 service instance with suggested nodes
calculate SLA violation ratio for the service
training? or evaluation?
training
evaluation
Deep Q-learning
get random output
calculate reward
observe reward
if reward is lower than the desired
Multilayer Network
output
QoS
9 feedback after execution
6 service instance with selected nodes

Fig. 4. QoS component

to improve the previous recommendations. This process is also shown in Fig. 3. First, we preprocess data by adding labels and normalizing values. Then, we train an ML model which will be used to generate recommendations based on the history of the network. After that, we update the KB by the new recommendations and finally we use those new recommendations to extract new actions that can be used by the online reasoning engine to recommend nodes for newly registered services. As with the previous component, the future steps include testing of the proposed design.

3.3 Quality of Service

As mentioned above, in order for the Resource Provider to make a decision on service instance execution, it will have to communicate with the QoS component.

This communication happens in the following step, after the Resource Provider suggests the list of nodes to be used for the service instance execution, the service instance is sent to the QoS component. The design of this block in shown in Fig. 4. The QoS component checks which of those nodes can actually be used for the service execution or if they have to be discarded in case they are not satisfying requirements to be considered as the potential solutions. In order to make this decision, the QoS component also gets informed about the existing Service Level Agreement (SLA) violations which are assumed to be stored in the Analytics component each time a service execution finishes. SLA management is out of the scope of our work with the assumption that SLA violations are detected and stored after the execution of the service. After making the decision on which nodes should be acceptable for the service instance execution, a modified service instance is returned to the Service Processor with the updated list of suitable nodes. The SLA violations are not considered by the Resource Provider component itself because the Resource Provider only takes into account the individual information from the nodes and the QoS analyses if the service as a global entity could be used for these devices.

The decision whether a certain resource can or cannot be used for a certain service, is based on the number of SLA violations that had occurred in previous executions of that specific service. With this information, the QoS Component uses reinforcement learning to allow or block the use of a specific node. The process that takes place in QoS component in order to decide whether the suggested nodes by the Resource Provider should be used, includes utilization of the number of service executions and the number of SLA violations. This information is used to calculate a ratio which is then passed on as the input for the Deep Q-learning (DQL) algorithm. Then, it has to be decided whether that input is taken for training or for evaluation (the decision process being described below). In the case of training, the DQL algorithm will initially get a random output, which determines which nodes are accepted. Based on the output, a reward is calculated following the next function:

$$r_t = \sum_{n=0}^{N} y_n(-2x_s + 1) + (1 - y_n)(x_s - 1), \tag{1}$$

where the N is the total number of nodes specified in the service instance, y_n is 1 when the node n is chosen, 0 otherwise, and x_s is the input ratio. The calculated reward is observed by the network and in case it is lower than a specific threshold, a new random output is generated and the process is repeated. When the reward is greater or equal than desired, the output is being used to modify the list of accepted nodes for the service instance. On the other hand, in the case of evaluation, the QoS component will directly ask the network about an optimal output for a specific input. How to decide if an input is taken for the training or for the evaluation is based on the quantity of already acquired knowledge in the network. For now, this decision is only based on a certain number of service executions.

While the QoS component could use the reward function without the need of using deep learning, the output would be only determined by that function, missing other non-trivial factors like the relation between the failure of the execution of a service and the nodes that were involved. For that reason, the proposed algorithm can be used to learn in every situation by taking random decisions and helping the optimization of the decision making process in the evaluation period. To be noted, at the moment the presented algorithm is a relatively simple version that could be used for testing of the proposed system. In the future, the reward function, the input, the output or how the decision to opt for a training or evaluation case is taken could change in order to improve the effectiveness of the algorithm.

3.4 Analytics

One of the most important SMS components which is used to update all the others is the Analytics component. It is responsible for gathering data generated from devices which allows it to offer an overview of network statistics. This component includes traffic prediction module which is used to enhance effectiveness of other modules by predicting traffic flows based on old statistics stored in the database. The need of automating the process of obtaining the analytics and the existence of datasets collected in this component opens the door for ML and AI to be implemented. Automation can dynamically extract insights from network statistics and implement the right algorithms with achieving a high performance results, instead of this task being handled by a developer, especially in the cases when patterns are not visible for a human being. The Analytics might include several features to provide a better view of the network such as device locations, device connections and signal coverage, but, we focused on gathering the data about the traffic and then generating insights to produce a real time visibility as one of the more interesting features. In this context, we studied only this feature of the Analytics, which analyzes the network traffic and leave the other features for the future research. In our system this feature is called Traffic prediction.

Traffic Prediction. In this paper, we focus on a service traffic prediction which we located in a F2C architecture as a part of the Analytics component. As a first requirement for this component to implement ML we need a dataset that represents the history of traffic demand at each instant of sampling, obtained from the SMS database. In our case, the goal is to study the temporal evolution of the traffic demand in a network, and to see how it can later be used to improve Analytics component of F2C system. It is necessary to use historical data which can be a real data or data that is modelled theoretically. We referenced several models that were used to analyze the traffic in different networks such as mobile cellular networks in order to generate out own dataset. [10] introduced a model to simulate the traffic variation for a base station in real cellular networks. The model used sinusoid superposition modelling method to describe the temporal traffic variation. [11] studied the network traffic in 10 data centers of different organization types (university, enterprise and cloud data centers). The study shows that the lognormal distribution can fit the time series of data center traffic. Thus, we use Eq. (2) to generate the mean values, then we use the lognormal distribution to generate traffic demands for each mean value.

$$Mean(t) = a_0 + \sum_{k=1}^{n} a_k \sin(w_k t + \phi_k) \tag{2}$$

In our case, $Mean(t)$ presents the total traffic demands in the data center, a_0 is the mean value of all traffic demands during 24 h, w_k is the frequency components of traffic, a_k and ϕ_k represent the amplitudes and phases, n is the number of frequency components. Table 1a summarizes the different values used to generate the mean values. As a result, we obtain the following equation:

$$Mean(t) = 100 + 70 \sin\left(\frac{\pi}{12} t + 3.11\right) + 30 \sin\left(\frac{\pi}{6} t + 2.36\right) \tag{3}$$

Time series prediction has been studied for a long time using traditional statistical techniques to solve forecasting problems. In the last two decades, recurrent neural networks proved to have good performance results in time series predictions due to their ability to capture short and long term dependencies. Our goal was to predict multiple future values based on a sequence of previous

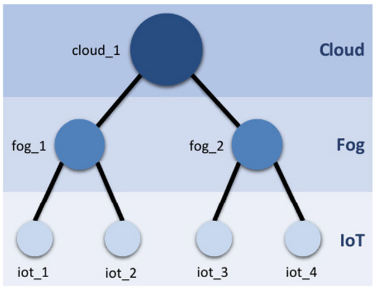


Fig. 5. Tested F2C architecture

demands. As we used recurrent neural network, we were able to create a multi-step forecasting model. This allowed prediction of all of the values in the time window using only one model. We generated data gathered in a two-month period to train and to test a Long Short Term Memory (LSTM) network. The data was then divided into 67% for training and 33% for testing.

Table 1. Experiment parameters and results

(a) Parameters

param.	*iot_1*	*iot_2*	*iot_3*	*iot_4*
n	2	3	3	3
w_1,w_2,w_3	$\frac{\pi}{12}, \frac{\pi}{6}, 0$	$\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{3}$	$\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{2}$	$\frac{\pi}{12}, \frac{\pi}{6}, \frac{\pi}{2}$
a_0	100	100	100	100
a_1,a_2,a_3	70,30,0	70,20,20	60,20,15	60,25,15
ϕ_1,ϕ_2,ϕ_3	3.11,2.36,0	5,1,1	7,5,2	1,1,1

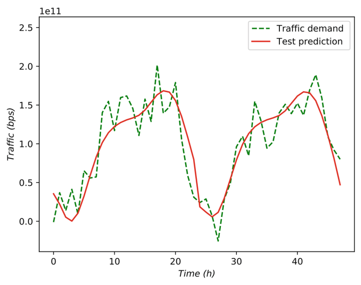
(b) Test Score (RMSE)

	smooth	**noisy**
iot_1	0.046	0.238
iot_2	0.074	0.242
iot_3	0.095	0.261
iot_4	0.087	0.257
fog_1	0.077	0.340
fog_2	0.186	0.376
cloud_1	0.125	0.529

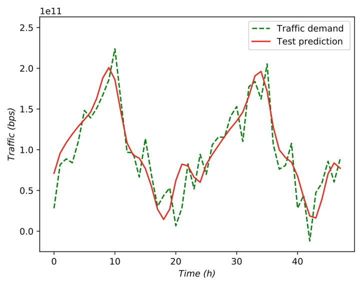
After this step, we used a Keras API running on the top of Tensorflow to obtain the forecast results. Generally, recurrent neural networks need a periodic data to be able to offer good forecasting results. As a result, the data was mapped into sequences of length 24 to be able to capture the data relationship during a whole day. The LSTM network has three layers: input, hidden and output layer. The model consists of an input layer, one hidden layer with 48 units and an output layer with hyperbolic tangent function as an activation function. We used mean square error as a loss function and the Root Mean Square Error (RMSE) to measure the accuracy. The RMSE is defined as follow:

$$RMSE = \sqrt{\frac{\sum_{k=1}^{n}(\hat{y}_k - y_k)^2}{n}} \tag{4}$$

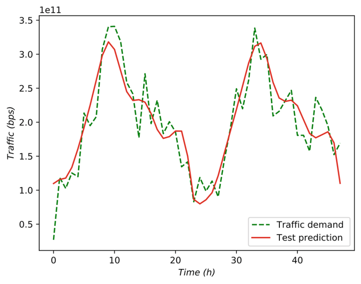
RMSE has the same unit as the data and it estimates the difference between true values and predictions.



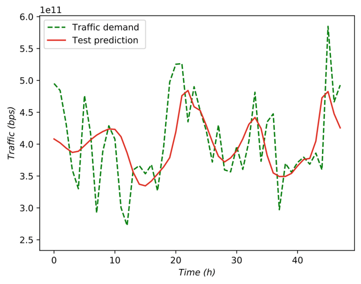
(a) iot_1



(b) iot_2



(c) fog_1



(d) cloud_1

Fig. 6. Traffic prediction results

Performance Evaluation. In this section, we evaluate the performance of the traffic predictor algorithm previously proposed running independently in each node of the F2C architecture shown in Fig. 5. We assume that traffic is generated by devices connected to the IoT nodes or by the IoT nodes themselves, and this traffic is sent to the cloud through the fog nodes. The IoT layer, we assumed that consists of nodes that present processing capable devices, which would allow ML to be implemented in this layer. As a result the distribution of traffic in higher network layers can be modeled as the sum of traffic flows coming from nodes in the lower layers. We assume there are no additional constraints (link capacity, node capacity, etc.), so we are able to send all the generated traffic to the cloud. In this paper, we evaluate the performance of LSTM with 4 different traffic flows generated by the four IoT devices using Eq. 2. Then, for instance, the node fog_1 receives together the traffic generated by the iot_1 and iot_2, and the cloud receives the traffic from fog_1 and fog_2. We evaluate the performance of the prediction algorithm for these three layers in two cases: with smooth artificially generated traffic without random noise and with the same shape but adding random noise.

The numerical results, shown in Table 1b, show that the RMSE is low when the traffic is predicted in the IoT layer without noise, but it is doubled when the noise is added. However, when the prediction is tested in the fog or in the cloud layer, RMSE increases in all cases being with noise the worst case and the cloud the layer with worse performance. Figure 6a and b show the traffic generated by the *iot_1* and *iot_2* devices for the case when there is random noise. Here, it can be seen how the prediction relatively well matches with the real traffic. When we add both traffics to the *fog_1* node, see in Fig. 6c, the LSTM network still can perform well but with worse performance compared with the previous case. Finally, when the traffic in the fog layer is added to the *cloud_1* node, the algorithm can not catch some of the periodical raises. This can be explained by the effect of noise which is accumulated from different traffic flows to make it difficult to the algorithm to differentiate between periods and noise. Since the traffic in the upper layers is the sum of traffics coming from different nodes and having a different periodicity, it results with a more complicated function. Also, each traffic has its own noise the sum of noises makes the traffic more random, so that the prediction become less efficient. That is why in the results we can see better prediction in the nodes close to the traffic generation.

4 Conclusion

With the interest in fog computing and architectural solutions that integrate fog and cloud, the focus is on developing and exploring new approaches and technologies, that would lead to significant improvements of these integrated F2C systems. With that in mind, in this paper, we proposed an architectural design of a service management component for a F2C system, and explored a ways how different machine learning algorithms could be used in different composite components of service management. In order to improve Analytics component we implemented LSTM network to evaluate the performance of the traffic prediction algorithm running independently in each node of the F2C architecture. The traffic generated by nodes representing IoT devices was observed for two cases, with and without noise added to the traffic. The results have shown that closer the prediction is to the source of generated data, the prediction results will be better in both cases. So the best prediction was achieved when it was performed in IoT layer without added noise, with the assumption that IoT layer consists of nodes with processing capabilities. Fog layer whose nodes were used as the aggregating points for multiple IoT generated traffic flows also performed well in terms of being able to predict close to real traffic. The worst traffic prediction was achieved in the cloud layer, which received aggregated flows from the fog layer. As a further step, QoS component improvement was implemented with a Deep Q-Learning algorithm, enabling it to make decisions whether a use of a certain node will be allowed or blocked, based on number of SLA violations that had occurred in previous executions of a specific service. The numerical results and improvement of decision making process for this algorithm are planned for the future. Additionally, in this paper, we propose the utilization of ML in components tasked with service classification and resource provisioning, with the implementation part as a goal for future work.

Acknowledgment. This work has been partially performed in the framework of mF2C project funded by the European Union's H2020 research and innovation programme under grant agreement 730929.

References

1. OpenFog Consortium. http://www.openfogconsortium.org/. Accessed Apr 2018
2. mF2C Project. http://www.mf2c-project.eu/. Accessed 20 Apr 2018
3. Papazoglou, M.P., van den Heuvel, W.J.: Web services management: a survey. IEEE Internet Comput. **9**(6), 58–64 (2005). https://doi.org/10.1109/MIC.2005.137
4. Amanatullah, Y., Lim, Y., Ipung, H.P., Juliandri, A.: Toward cloud computing reference architecture: cloud service management perspective. In: International Conference on ICT for Smart Society, Jakarta, pp. 1–4 (2013). https://doi.org/10.1109/ICTSS.2013.6588059
5. Guo, J., Chen, I.R., Tsai, J.J.P., Al-Hamadi, H.: A hierarchical cloud architecture for integrated mobility, service, and trust management of service-oriented IoT systems. In: 2016 Sixth International Conference on Innovative Computing Technology (INTECH), Dublin, pp. 72–77 (2016). https://doi.org/10.1109/INTECH.2016.7845021
6. Castro, A., Villagra, V.A., Fuentes, B., Costales, B.: A flexible architecture for service management in the cloud. IEEE Trans. Netw. Serv. Manag. **11**(1), 116–125 (2014). https://doi.org/10.1109/TNSM.2014.022614.1300421
7. Agyemang, B., Xu, Y., Sulemana, N., Liu, N.: Resource-oriented architecture toward efficient device management and service enablement. In: 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, pp. 2561–2566 (2017). https://doi.org/10.1109/SMC.2017.8123010
8. Yin, Y., Wang, L., Gelenbe, E.: Multi-layer neural networks for quality of service oriented server-state classification in cloud servers. In: 2017 International Joint Conference on Neural Networks (IJCNN), 14–19 May 2017. https://doi.org/10.1109/IJCNN.2017.7966045
9. Hwang, J., Liu, G., Zeng, S., Wu, F.Y., Wood, T.: Topology discovery and service classification for distributed-aware clouds. In: 2014 IEEE International Conference on Cloud Engineering (IC2E), March 2014. https://doi.org/10.1109/IC2E.2014.86
10. Zhang, X., Wang, S., et al.: An approach for spatial-temporal traffic modeling in mobile cellular networks. In: Teletraffic Congress (ITC 27). IEEE (2015)
11. Maltz, D.A., Benson, T., Akella, A.: Network traffic characteristics of data centers in the wild. In: IMC '10 Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement. ACM, New York (2015)