Smart SDN Management of Fog Services

Piotr Fröhlich, Erol Gelenbe and Mateusz P. Nowak
Institute of Theoretical and Applied Informatics (IITIS-PAN)
Polish Academy of Sciences (PAN)
IITIS-PAN, ul. Baltycka 5
44100 Gliwice, Poland

***Abstract*—We present a smart Service Manager whose role is to direct user requests (such as those coming from IoT devices) at the edge towards appropriate servers where the services they request can be satisfied, when services can be housed at different Fog locations, and the system is subject to variations in workload. The approach we propose is based on using an SDN controller as a decision element, and to incorporate measurement data based machine learning that uses Reinforcement Learning to make the best choices. The system we have developed is illustrated with experimental results on a test-bed in the presence of time-varying loads at the servers. The experiments confirm the ability of the system to adapt to significant changes in system load so as to preserve the QoS perceived by end users.**

***Index Terms*—QoS, Edge and Fog Computing, IoT, Software Defined Networks, Machine Learning, Cognitive Packet Network, Random Neural Networks**

I. Introduction

Fog computing extends the concept of the Cloud [1] to allow edge devices to take over substantial computation, storage and networking activities, so as to facilitate the operation of services between edge devices and Cloud data centers [2]. Therefore it is particularly suited as an approach to manage services and tasks in the Internet of Things (IoT) [3], [4].

A high degree of service virtualization is a characteristic of many computing platforms, and the Fog infrastructure will have computing nodes at its disposal that run virtualized services that satisfy the requests of the clients connected to the network. Thus the distribution or location of Fog nodes/servers in a network, and the placement of services on the Fog servers are key issues. Since most networks may have a large variability of workload over time, the configuration of Fog servers and services cannot be static, and a dynamic approach is needed to adapt it to changing situations. Such issues are not specific to the Fog and they have long been studied in the context of distributed and networked computer systems [5], [6], [7], [8], [9]. However, the nature of the servers and lightweight edge devices in the Fog call for simple dynamic algorithms which do not impose excessive decision making overhead. Thus in this paper we develop a fast decision algorithm for directing requests that originate at different devices towards the multiple servers where services are located, without causing significant additional workload for the edge devices and the servers, and we exploit the presence of Software Defined Network (SDN) controllers in their midst so as to provide a Service Management function in addition to their usual packet routing activities.

The dynamics of Fog substrate configuration can be considered at different levels. The most advanced, but perhaps the most costly approach is to migrate services between Fog nodes. This solution is worth using if there are very significant changes in network and server load, and in the memory ocupancy of the servers. Indeed service migration takes time and bandwidth, also limiting the availability of services during migration and temporarily reducing the resources available to end users.

In IoT networks in particular [3], due to the relatively steady nature of monitoring and actuating on cyberphysical infrastructures, significant changes in network usage are relatively infrequent. Therefore, in most cases it suffices to locate each service in a few replicate locations and to optimally select the location of a service to which a given client's request should be addressed. If needed, different instances of a service can also be activated on-the-fly when repliacs of a service are installed at different system nodes. However, replicating services also raises the issue of consistency of the data and it may introduce additional slowdown and overhead if consistency control algorithms need to be used [10].

The optimization of such systems with respect to Quality of Service (QoS) and Energy Consumption using queueing theoretica techniques was considered in earlier work [11]. Similar problems related to the allocation of tasks to servers in the Cloud were considered in [12] using Reinforcement Learning (RL) [13] and other Machine Learning (ML) methods such as Deep Learning [14].

In this paper we discuss the problem of allocating a user's request for access to a given service s which is located at several ($N(s) > 1$) different servers or Fog nodes. The approach we take is based on defining a relevant cost or Goal Function which includes the measured QoS, and then making decisions in real time regarding the choice of the service's location, using RL to optimize the user's perceived QoS. We implement the proposed algorithm as a Service Manager platform which is transparent to the end user and is installed in a Software Defined Network (SDN) controller. We illustrate the performance of our system with experiments which show its effectiveness in the presence of dynamic time-dependent changes in system workload.

In the sequel, in Section II we first present a formalisation of the optimization problem for the selection of an instance of a service for the requst made by some user, when multiple instances of several services are located at the nodes of a

Fog platform. Section III discusses the mathematically based ML method that we use, and in Section III-B we detail the reinforcement learning algorithm that is at the heart of the system that we have designed and tested. Section IV presents the specific example that we have experimentally tested in this paper, where the Service Manager is in charge of selecting a particular location where a service request formulated by some user will be executed with the objective of optimizing the resultin QoS. The experimental results that we present show how the service requests are dynamically allocated, and re-allocated to another server if a given server becomes overloaded due to excessive workload. The final section is devoted to drawing some conclusions and suggesting directions for further work.

II. A Representation of the Decision System

We consider a system consisting of N nodes $\{1, \ ... \ , N\}$ where the nodes can be connected via an underlying multi-hop Internet topology. Thus the N nodes can be viewed as an overlay networnodes of Fog servers which are also access points for IoT devices or gateways that are attached to the overlay nodes. Any two Fog nodes can communicate and transfer data and tass to each other.

Services, such as data storage systems, named data servers, content providers or services that execute tasks, are located at these Fog nodes. Service requests are formulated by users, and can then be directed towards one of these nodes by the "Fog Manager" (FM) which is a decision system that mey reside at each of the nodes, or which may itself reside at some other node. Forthe purposes of this paper, we do not dwell on where the FM resides and we viewed itas some form of transparent instantaneous decision system.

The general problem we formulate is about placing the set of users U and the set of services S at the various nodes or locations. Some user u at location $l(u) \in \{1, \ ... \ , N\}$ generates requests $R(u)$ to some service s so that $R(u) = s$. The location of s will be denoted $l(s) \in \{1, \ ... \ , N\}$. Generally users may be mobile, but will make a request from a specific location. On the other hand, a service s may be duplicated at a set of locations $L(s) \subset \{1, \ ... \ , N(s)\}$.

The request from u to $s \in L(R(u))$ is satisfied with after some transfer delay $T(l(u), l(R(u))$ which depends on the nodes where the user and service are located, and on the currently used network paths between them. Furthermore, the queueing plus service delay $D(.)$ needed to satisfy the request will also depend on the node $l(R(u)$ that services the request, and on its current load that we denote by $K(l(R(u)))$. Thus we will have some non-linear dependency $D(K(l(R(u)))$, to which we should add the load-dependent local delay at the node where u is connected, which we will denote $d(K(l(u)))$.

Therefore when a user u makes a request for a service $s = R(u)$ the purpose of the FM s to try to minimize an objective function of the form:

$$\begin{aligned} G(u, l(R(u))) \ &= \ T(l(u), l(R(u))) + D(K(l(R(u))) \\ &+ d(K(l(u))) + \alpha I(u, l(R(u))) \\ &+ \beta \mathbf{E}(u, R(u)), \end{aligned} \quad (1)$$

where $\alpha, \ \beta \geq 0$ are constants, and:

- $I(.)$ refers to a on-negative numerical value that characterizes the "insecurity" of having user u access service $R(u)$ at location $l(R(u))$. We note that this insecurity can actually be due to the user or its sensitivity, rather than the location, or it can be interpreted as depending on some risks or attacks that are related to the location $l(R(u))$, which is the more likely case.
- $E(.)$ refers to the resulting energy consumption, and:

$$\begin{aligned} \mathbf{E}(u, R(u)) \ &= \ E(l(u), l(R(u))) + E(K(l(R(u))) \\ &+ \ E(K(l(u))). \end{aligned}$$

The minimization of $G(u, R(u))$ will be carried out over all possible locations $l(R(u)) \in L(R(u))$. When we are free to instantiate the service $s = R(u)$ on any of the servers of the system, then we will obviously have $L(R(u)) = \{1, \ ... \ , N\}$.

The minimization of $G(u, R(u))$ is the optimization problem that is discussed in this paper, and we would tend to allocate the request $s = R(u)$ of user u to the node:

$$i^*(u, s) = \arg\min\{G(u, i) : \ s.t. \ i \in L(R(u))\} \ . \quad (2)$$

More restricted cases of this problem have been considered in earlier work. In [12], the services are duplicated at all the nodes, and requests emanate from a single node and are then dispatched to any one of the nodes using an RNN based Reinforcement Learning scheme. Other work [15] uses an RNN based algorithm that considers both remote and local nodes so that the transfer of requests to remote nodes incurs a communication delay plus a processing delay, while local nodes have a congestion based queueing delay plus a request processing time. Note that in (2) each of the terms $D(K(l(R(u)))$ and $d(K(l(u)))$ can include both a queueing delay waiting for service at the node, and a service time.

III. The Random Neural Network (RNN) and Reinforcement Learning

Because the parameters in the Goal function can only be learned or estimated through measurement over some period of time, we propose a machine learning approach, and we first introduce the neural network model that will be used, which is recurrent, i.e. it contains feedback between its nodes. In fact, its adjacency graph is a fully connected directed graph on identical to the topology of the possible IP connections between nodes in the real system.

The neural network model we use is the recurrent Random Neural Network (RNN) [16] because of two of its important mathematical properties: it has a convenient closed form analytical solution in "product form", and it has an unique numerical solution despite its recurrent non-linear structure. Thus for a given set of input parameters it is guaranteed to provide a unique state and output value. We will also have one distinct neuron for each of the N distinct nodes or servers where services may be placed, and the RNN will be used to compute the node to which a specific service request must be directed.

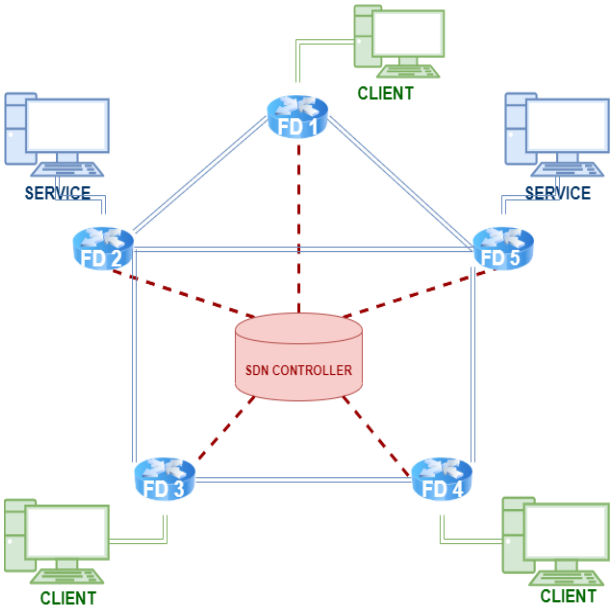

Fig. 1. Topology of a 5-node packet network with 6 inter-node links, and 5 attached servers, two of which support services and the others support end users. The SDN controller comunicates with every node and acts not just to establish network paths, but also to decide which service location or instance will be used by the service requests. This system is used as a test-bed for the experiments reported in this paper.

An N neuron RNN is a probabilistic dynamical system whose state is represented by the vector of non-negative integers $K(t) = (K_1(t),\ ...\ , K_N(t))$ at time $t \geq 0$, where $K(t)$ is a vector random process. A particular value taken by $K(t)$ is denoted by the deterministic vector $k = (k_1,\ ...\ k_N)$. $K_i(t)$ represents the "voltage" or potential of neuron i. The neurons are interconnected via excitatory and inhibitory weights that are denoted by $W^+_{ij} \geq 0$, $W^-_{ij} \geq 0$, respectively. These weights can be viewed as rates of spiking from any neuron i to any neuron j.

Each excitatory spike sent from i and arriving at time t to j will increase the value of $K_j(t)$ by $+1$, i.e. its effect will be $K_j(t^+) = K_j(t) + 1$. Similarly, each inhibitory spike sent from i to j at time t will have the following efffect $K_j(t^+) = max[K_j(t) - 1, 0]$. However a neuron i cn only send out spikes if its potential is positive, i.e. when $K_i(t) > 0$. Furthermore, when neuron i sends a spike to neuron j, then its own potential drops by 1, i.e. $K_i(t^+) = k_i(t) - 1$.

The key theorem concerning the RNN [16] states that:

$$\lim_{t\to\infty} Prob[K(t) = k] = \prod_{i=1}^{N} q_i^{k_i}(1 - q_i),\ \ where \quad (3)$$

$$q_i = \frac{\Lambda_i + \sum_{j=1}^{N} q_j W^+_{ji}}{\lambda_i + \sum_{j=1}^{N}[W^+_{ij} + W^-_{ij}] + \sum_{j=1}^{N} q_j W^-_{ji}}. \quad (4)$$

We also use r_i to denote the quantity $r_i = \sum_{j=1}^{N}[W^+_{ij} + W^-_{ij}]$, and we call it the "total firing rate" of neuron i.

Note that each decision is user and service dependent, and different users may have different locations in the network. Therefore in general we may have a distinct RNN for each user and service, and we can write:

$$q_i(u,s) = \frac{\Lambda_i(u,s) + \sum_{j=1}^{N} q_j(u,s) W^+_{ji}(u,s)}{\lambda_i(u,s) + r_i(u,s) + \sum_{j=1}^{N} q_j W^-_{ji}(u,s)}, \quad (5)$$

where $r_i(u,s) = \sum_{j=1}^{N}[W^+_{ij}(u,s) + W^-_{ij}(u,s)]$, is the "total firing rate" of neuron i.

Let $i^*(u,s) = \arg\max i\{q_i(u,s)\}$: we will consider that $i^*(u,s)$ is the node that is preferred by the decision algorithm to select the location of the service $s = R(u)$ requested by user u; hence it is in some sense the node that is estimated to provide the best performance to the current service request from user u for the service $s = R(u)$.

A. Initialisation of the Recurrent RNN

Before any data has been gathered, and before the RNN weights are updated using the reinforcement learning algorithm that we describe in the following section, the RNN weights should be updated in a manner that makes all the $q_i(u,s) = 0.5$ to represent a situation where all possible choices are equally likely, and all weights are identical, i.e.

$$\begin{aligned} w &= W^+_{ij}(u,s) = W^-_{ij}(u,s), \quad (6)\\ \lambda &= \Lambda_i(u,s) = \lambda_i(u,s),\ \forall\ i,\ j,\ u,\ s, \end{aligned}$$

which will yield the equation:

$$0.5 = \frac{\lambda + 0.5Nw}{\lambda + 2.5Nw},\ or\ \lambda = 1.5Nw. \quad (7)$$

Thus to obtain $q_i(u,s) = 0.5$, we can set w to any value, as long as we also set $\lambda = 1.5Nw$.

B. The Reinforcement Learning Algorithm

The Goal function G or G' of (2) or (**??**) will be used with the RNN and a Reinforcement Learning (RL) algorithm to optimize the system. The objective is to choose the best node i where the service $s = R(u)$ requested by user u should be instantiated or located. We first define the *Reward* $R(u,s) = G(u,s)^{-1}$ or $R(u,s) = G'(u,s)^{-1}$ which must be maximized when the Goal is minimized. Successive values of $R(u,s)$ are measured, or measured and estimated. For instance, transfer times between the location of u and the different nodes in the network can be measured, and they do not depend on actually executing a user request for a service. Similarly, the execution time of a service at different locations for other users u', other than the actual user u, can be used to estimate $D(K(l(R(u)))$, while $d(K(l(u)))$ can be estimated by measuring the performance related to the local node where u is residing.

Successive values of the "reward" $R_l(u,s) = G_l(u,s)$, $l = 1, 2,\ ...$ will be obtained from the successive measured Goal values $G_l(u,s)$, $l = 1,\ 2,\ ...$ that are brought back by SPs and are used them compute "historical value" of the reward:

$$T_l(u,s) = \delta * T_{l-1}(u,s) + (1-\delta) * R_l(u,s), 0 < \delta < 1, \quad (8)$$

where $0 < \delta < 1$ is a responsiveness parameter that determines the importance of past historical values. Setting it to a high

value will prevent the RNN from taking hasty decisions. The RNN weights are then updated as follows.

First save the current values of the sum of the weights $r_i(u,s) = \sum_{j=1}^{N}[W_{ij}^{+}(u,s) + W_{ij}^{-}(u,s)]$. Let k be the most recent selected "best" choice of the location for service s with regard to user u, i.e. $k = i^*(u,s)$ or $k = I^*(u,s)$. Then:

$$If \quad R_l(u,s) >= T_{l-1}(u,s) \quad then \quad for \quad j \neq k: \tag{9}$$

$$\begin{aligned} \forall i \neq k: \qquad & W_{ik}^{+}(u,s) \leftarrow W_{ik}^{+}(u,s) + R_l(u,s), \\ & W_{ij}^{-}(u,s) \leftarrow W_{ij}^{-}(u,s) + R_l(u,s), \end{aligned} \tag{10}$$

$$If \quad R_l(u,s) < T_{l-1}(u,s) \quad then \quad for \quad j \neq k: \tag{11}$$

$$\begin{aligned} \forall i \neq k: \qquad & W_{ik}^{-}(u,s) \leftarrow W_{ik}^{-}(u,s) + R_l(u,s), \\ & W_{ij}^{+}(u,s) \leftarrow W_{ij}^{+}(u,s) + R_l(u,s). \end{aligned} \tag{12}$$

After these updates, a normalization is carried out for all the weights, preventing them from constantly increasing:

$$W_{ij}^{+}(u,s) \leftarrow W_{ij}^{+}(u,s) \frac{r_i(u,s)}{\sum_{j=1}^{N}[W_{ij}^{+}(u,s) + W_{ij}^{-}(u,s)]}, \tag{13}$$

$$W_{ij}^{-}(u,s) \leftarrow W_{ij}^{-}(u,s) \frac{r_i(u,s)}{\sum_{j=1}^{N}[W_{ij}^{+}(u,s) + W_{ij}^{-}(u,s)]}. \tag{14}$$

Now with these updated values of the weights, we compute all the $q_i(u,s)$ using the system of equations (5), and obtain the new value of the "best location":

$$i^*(u,s) = \arg\max\{q_i(u,s)\}. \tag{15}$$

IV. Service Duplication at Several Locations

In our current implementation and experiments, we use a simpler load independent form of the Goal Function (2), where we aggregate the network transfer time and service delay into a single term:

$$Q(u, l(R(u))) = T(l(u), l(R(u))) + D(u, l(R(u))), \tag{16}$$

because these two quantities are measured in our experiments as one single value, which use as the Goal for the RL algorithm:

$$\begin{aligned} G(u, l(R(u))) \quad = \quad & Q(u, l(R(u)) + \alpha I(u, l(R(u)) \\ & + \beta \mathbf{E}(u, l(R(u))). \end{aligned} \tag{17}$$

The experimental platform on which these ideas have been implemented and tested is represented in Figure 1 where the five network nodes can be used to support either users or services. In this case we see that three nodes support users, while two nodes support services, and the six links that exist between nodes are also explicitly shown. Both the services and the users are in fact on separate machines which are connected as shown to the network nodes.

A. Network Level Path Control

The system, both for network routing and for accessing services by specific users, is run by a SDN controller [17], [2], [18], [19], via a switch, which is connected to each of the five network nodes as shown in Figure 1. The DN controller uses OpenFlow Version 1.2-1.5 [20].

The SDN system in our test-bed has been extended using the "cognitive packet routing algorithm" described in [21] to conduct smart measurements of network delays using "smart packets" (SP) so as to set up network paths in a manner that minimizes source-to-destnation packet delays, similar to the approach taken in [22]. The SDN controller checks the network state each 5 seconds, and network paths can be changed at those times if significantly better paths are found that exceed the previous measured source-to-destination delay by a given threshold over 30%.

B. Service Management

The SDN controller in our system is also in charge of the allocaton of a user u's requests $R(u)$ to the locations $l(R(u))$. where the requested service is resident and may be satisfied.

Within the SDN controller, for each user-service pair (u,s), we install a RNN which has a number of RNN nodes identical to the number of location where the service can be found, which we denote $N(s)$. For instance, in Figure 1 we have $N(s) = 2$.

The weights of the RNN for the pair (u,s) are updated using Reinforcement Learning as described in Section III-B, based on measurements sent to the SDN controller by each user, and specifically the user's own perceived average total response time, from the instant when the request $R(u)$ is sent by u to the location $l(R(u))$, to the instant when the successful response was received by the user u, which corresponds to the quantity $Q(u, l(R(u)))$ previously defined. Thus these experiments are based on learning using:

$$G(u, l(R(u))) = Q(u, l(R(u))), \tag{18}$$

without using either the "insecurity factor" or the energy consumption. The RNN weights are updated according to the algorithm in Section III-B, where the choice of the optimum location from the values $q_i(u,s)$ for $1 \leq i \leq N(s)$.

From the user point of view this solution is completely transparent. The user u is given a configuration file which includes an IP address and the port $(IP, Port)$ on which the service s can be found. Note that this is an IP address which is unavailable at the network level. Each time u wants to connect to s, it connects to $(IP, Port)$. On the edge node where u is connected, the SDN controller changes $(IP, Port)$ to the IP address of the real location $l(s)$ of s. When the service ends and the resulting reply goes back from the location of the service to the user, the real IP address is changed back to the original "dummy' IP address provided in the configuration file.

C. Experimental Results

The purpose of our experiments were to show the ability of the algorithms that we have designed and implemented,

to provide rapid adaptation to changes in the measured QoS at the nodes. The way in which the QoS has been varied, is by placing at each of the nodes a specific additional program which overloads the CPU at each of the two servers where the service is located.

We have conducted numerous experiments on this test-bed, with and without the Service Manager being turned on. In these experiments the users that are attached to servers in nodes $1,\ 3,\ 4$ generate a sequences of successive requests for the service s which is located at the servers attached to nodes $2,\ 5$ and we show the resulting measured response times, with the Service Manager being off or turned on in Figure 2.

In the upper curve we see the effect of a sudden increase in workload due to an addditional external load on the server attached to node 5 of Figure 1, when the Service Manager is not use: the user experiences a sudden increase in its perceived total response time as soon as the workload at server is increased.

On the lower curve of the same figure we see results from another experiment when the Service Manager is turned on: when the load at node 5 suddenly increases, first the response time for the user requests increases, but after a transient of approximately $2,000$ milliseconds, the total average response perceived by the user drops to normal, despite the increase of workload at the server attached to node 5 because the Service Manager is enabled and updates the IP address that the user should access to node 2.

In the next Figure 3, the user's service is initially being served at the server attached to node 5. Then at roughly $40,000$ milliseconds, the server attached to node 5 is overloaded by an external load (other than the service) and the user's perceived response time rises dramatically. But the server attached to node 2 is still free of extra load, and the SM transfers the requests to the server attached to node 2. At time $100,000$ milliseconds exactly the opposite occurs and now the server attached to node 5 becomes overloaded, and after some delay the SM transfers the service requests back to the previous serever. Agin around time $270,000$ milliseconds we go back to the inital situation. We see that the Service Manager whih has been kept on throughout this experiment, has been effective in preserving the QoS perceived by the end user in the presence of these changes in additional load at the different servers.

V. Conclusions

This paper has presented a novel control algorithm and implementation for a Service Manager that advises end user on the best location for the services that they may need. The approach uses machine learning, namely Reinforcement Learning with Random Neural Networks, which attempts to offer the best decision based on on-line measurements which are used for learning.

The general framework for our approach is presented via an objective or Goal Function which can include factors such as Quality of Service, as well as Energy Consumption and Security (or insecurity). We have also detailed the algorithms used.

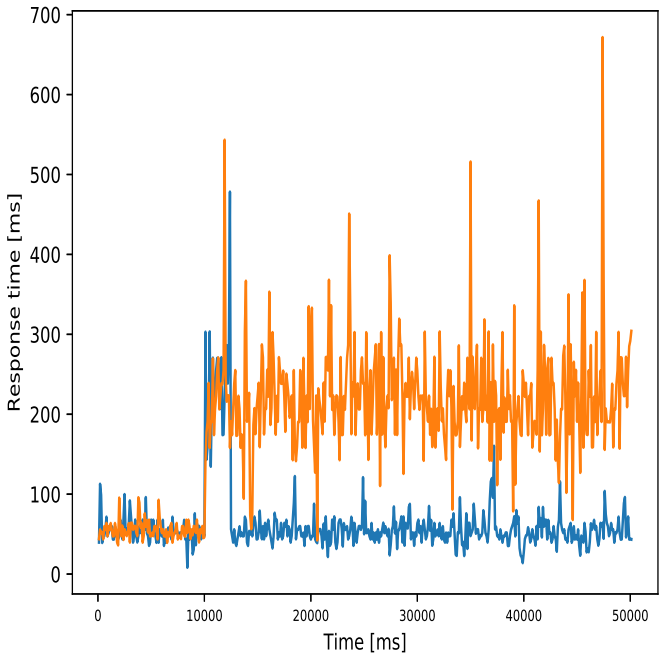

Fig. 2. The upper curve show the high increase in response times perceived by the end user when Service Manager is disabled, and the external (not user based) workload at the server attached to node 5 is suddenly increased. The lower curve shows the same experiment when the Service Manager is enabled: after a brief transitory period, the user's measured response time drops back to "normal" because the Service Manager as redirected the user towards the server attached to node 2.

An experimental test-bed that uses an SDN controller has been implemented to incorporate these ideas ina practical setting. The operation of this test-bed and the Service Manager that we ave designed has been illustrated with some experiments. These experiments have not yet included the security and energy optimization aspects, but have clearly illustrated the ability of our system to adapt in real time to changes in load at different servers that support services that are needed by a user,.

Future work will continue the study to include specifically the effects of energy consumption and security. We also plan to present further results where multipe services compete for overall distributed system resources so that we may be able to investigate the system's ability to adapt in the presence of competing end-users and multiple services.

References

[1] R. Buyya et al., "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 105:1–105:38, 2019. [Online]. Available: https://doi.org/10.1145/3241737

[2] A. Levin, K. Barabash, S. G. Y. Ben-Itzhak, and L. Schour, "Networking architecture for seamless cloud interoperability," in *2015 IEEE 8th International Conference on Cloud Computing, New York, NY*, 2015, pp. 1021–1024.

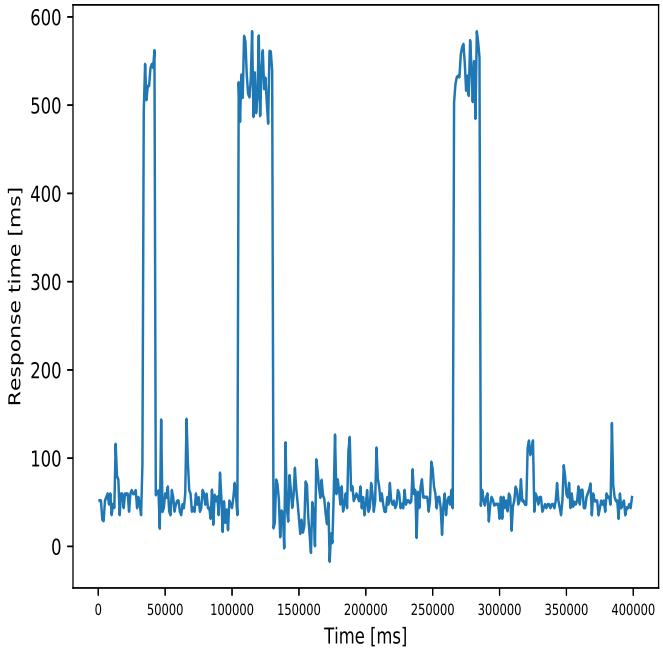

Fig. 3. This figure shows the results of an experiment where the response time to service requests is measured at the user end is plotted against elapsed time for a large number of successive service requests. The Service Manager (SM) is turned on throughout the experiment, and initially the service requests are being assigned to the server attached to node 5. At roughly $40,000$ milliseconds after the start of the experiment, the response time rises steeply because an external load has been imposed on the server attached to node 5, and then drops because the SM has transferred the user's requests from the server at node 5 to the server at node 2. At roughly $100,000$ milliseconds, the overload at the server at node 5 is turned off, and a similar overload is turned on at the server at node 2: again we observe a high increase in measured response time and then a drop to "normal" because the SM has transferred the request to the server attached to node 5. A similar switch occurs in the other direction at roughly $270,000$ milliseconds showing that the SM acts appropriately using the algorithm described in this paper.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, August 2012, p. 13–16.

[4] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Profit-aware application placement for integrated fog-cloud computing environments," *J. Parallel Distrib. Comput.*, vol. 135, pp. 177–190, 2020. [Online]. Available: https://doi.org/10.1016/j.jpdc.2019.10.001

[5] C. Kim and H. Kameda, "An algorithm for optimal static load balancing in distributed computer systems," *IEEE Trans. Computers*, vol. 41, p. 381–384, 1992.

[6] H. Topcuouglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distributed Systems*, vol. 13, no. 3, p. 260–274, 2002.

[7] X. Zhu, X. Qin, and M. Qiu, "Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE Trans. Computers*, vol. 60, no. 6, p. 800–812, 2011.

[8] W. Tian, Y. Zhao, Y. Zhong, M. Xu, and C. Jing, "A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters," pp. 311–315, 2011.

[9] Z. Zhang and X. Zhang, "A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation," in *Proc. 2nd Int. Conf. Industrial Mechatronics Automation*, vol. 2, 2010, p. 240–243.

[10] E. Gelenbe and K. C. Sevcik, "Analysis of update synchronization for multiple copy data bases," *IEEE Trans. Computers*, vol. 28, no. 10, pp. 737–747, 1979. [Online]. Available: https://doi.org/10.1109/TC.1979.1675241

[11] E. Gelenbe and R. Lent, "Energy-qos trade-offs in mobile service selection," *Future Internet*, vol. 5, no. 2, pp. 128–139, 2013. [Online]. Available: https://doi.org/10.3390/fi5020128

[12] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *IEEE Trans. Cloud Computing*, vol. 6, no. 1, pp. 33–45, 2018. [Online]. Available: https://doi.org/10.1109/TCC.2015.2474406

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2nd Ed., 2018.

[14] Y. Yin, "Deep learning with the random neural network and its applications," *CoRR*, vol. abs/1810.08653, 2018. [Online]. Available: http://arxiv.org/abs/1810.08653

[15] L. Wang, O. Brun, and E. Gelenbe, "Adaptive workload distribution for local and remote clouds," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016, pp. 3984–3988.

[16] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," *Neural Computation*, vol. 1, no. 4, pp. 502–510, 1989.

[17] J. Mambretti, J. Chen, and F. Yeh, "Next generation clouds, the chameleon cloud testbed, and software defined networking (sdn), 2015 international conference on cloud computing research and innovation (icccri)," pp. 73–79, 2015.

[18] S. Bera, S. Misra, and A. V. Vasilakos, *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, 2017.

[19] S. Basterrech and G. Rubino, "A tutorial about random neural networks in supervised learning," in *Neural Network World*, vol. 25, no. 5, 2016, pp. 457–499.

[20] "Home page of onosproject - open source sdn controller." [Online]. Available: https://onosproject.org

[21] E. Gelenbe, "Steps toward self-aware networks," *Communications of the ACM*, vol. 52, no. 7, pp. 66–75, 2009.

[22] F. François and E. Gelenbe, "Towards a cognitive routing engine for software defined networks," in *2016 IEEE International Conference on Communications, ICC 2016, Kuala Lumpur, Malaysia, May 22-27, 2016*, 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICC.2016.7511138