Auto-Scaling Network Service Chains Using Machine Learning and Negotiation Game

Sabidur Rahman, *Graduate Student Member, IEEE*, Tanjila Ahmed, *Graduate Student Member, IEEE*, Minh Huynh, *Member, IEEE*, Massimo Tornatore, *Senior Member, IEEE*, and Biswanath Mukherjee, *Fellow, IEEE*

***Abstract*—Network Function Virtualization (NFV) enables Network Operators (NOs) to efficiently respond to the increasing dynamicity of network services. Virtual Network Functions (VNFs) running on commercial off-the-shelf servers are easy to deploy, update, monitor, and manage. Such virtualized services are often deployed as Service Chains (SCs), which require in-sequence placement of computing and memory resources as well as routing of traffic flows. Due to the ongoing migration towards cloudification of networks, the concept of auto-scaling which originated in Cloud Computing, is now receiving attention from networks professionals too. Prior studies on auto-scaling use measured load to dynamically *react* to traffic changes. Moreover, they often focus on only one of the resources (e.g., compute only, or network capacity only). In this study, we consider three different resource types: compute, memory, and network bandwidth. In prior studies, NO takes auto-scaling decisions, assuming tenants are always willing to auto-scale, and Quality of Service (QoS) requirements are homogeneous. Our study proposes a negotiation-game-based auto-scaling method where tenants and NO both engage in the auto-scaling decision, based on their willingness to participate, heterogeneous QoS requirements, and financial gain (e.g., cost savings). In addition, we propose a *proactive* Machine Learning (ML) based prediction method to perform SC auto-scaling in dynamic traffic scenario. Numerical examples show that our proposed SC auto-scaling methods powered by ML present a win-win situation for both NO and tenants (in terms of cost savings).**

***Index Terms*—Auto-scaling, service chains, virtual network functions, machine learning, negotiation game, cost savings, QoS, resource disaggregation, edge datacenters.**

Manuscript received July 27, 2019; revised December 20, 2019 and April 11, 2020; accepted May 13, 2020. This work was supported by NSF Grant No. 1716945. The associate editor coordinating the review of this article and approving it for publication was K. Xue. *(Corresponding author: Sabidur Rahman.)*

Sabidur Rahman and Biswanath Mukherjee are with the Department of Computer Science, University of California at Davis, Davis, CA 95616 USA (e-mail: krahman@ucdavis.edu; bmukherjee@ucdavis.edu).

Tanjila Ahmed is with the Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA 95616 USA (e-mail: tanahmed@ucdavis.edu).

Minh Huynh is with AT&T Labs, Florham Park, NJ 07932 USA (e-mail: mahuynh@ucdavis.edu).

Massimo Tornatore is with the Department of Electronics and Information, Politecnico di Milano, 20133 Milano, Italy, and also with the Department of Computer Science, University of California at Davis, Davis, CA 95616 USA (e-mail: mtornatore@ucdavis.edu).

Digital Object Identifier 10.1109/TNSM.2020.2995900

I. Introduction

NETWORK functions, such as those implemented in firewall, switch, router, Customer Premises Equipment (CPE), etc. have been traditionally deployed on proprietary hardware equipment, referred to as "middleboxes". This has made equipment upgrade, deployment of new features, and maintenance complex and time consuming for Network Operators (NO). Recent migration towards Network Function Virtualization promises faster service deployment and flexible management [1]. Virtual Network Functions (VNFs) allow us to use Commercial-Off-The-Shelf (COTS) hardware to replace costly vendor hardware. VNFs can be hosted as virtual machine (VM) instances inside cloud datacenters (DCs), or in edge datacenter (Edge-DCs) locations such as smaller metro datacenters (Metro-DCs), telecom Central Office Re-architected as a Datacenter (CORD) [2], etc.

Network functions are often placed in an ordered sequence to process traffic flows resulting in *Service Chains* (SCs) [3]–[6]. A *Service Chain* involves a set of specific Network Functions (NFs) to be traversed, and a certain amount of network bandwidth to route the traffic through them. Fig. 1 shows two example service chains (SC_1 and SC_2) in a geo-distributed DC/Edge-DC scenario. SC_1 illustrates a service chain which traverses nodes 6, 3, 2, and 1, where three network functions are hosted (VMs NF_1 and NF_2 at node 6, and NF_3 at node 2). In addition to the computing and memory resources located at nodes, the NO allocates necessary network bandwidth capacity (N) to maintain Quality of Service (QoS) as per the tenant's request. Similarly, traffic flow for SC_2 traverses nodes 10, 14, 13, and 12, with NF_1 at node 10 and NF_3 at node 13.

Often a SC is placed according to a tenant's request. Traditionally, a tenant leases enough resources from the NO to support the tenant's peak load. For a static allocation of resources, after the NO finalizes the initial placement of a SC with resources required by the tenant, the allocated (computing, memory, and bandwidth) resources remain unchanged for the rest of the SC's lifetime [6]. But, in practice, the traffic demand flowing through the SCs (for the respective tenant) can vary over time, leading to a dynamic resource-allocation scenario. This idea of auto-scaling (i.e., varying the number of allocated resources automatically based on load) originated in cloud computing [7].

Cloud service providers implement auto-scaling capability for computational resources, so a tenant pays leasing cost only

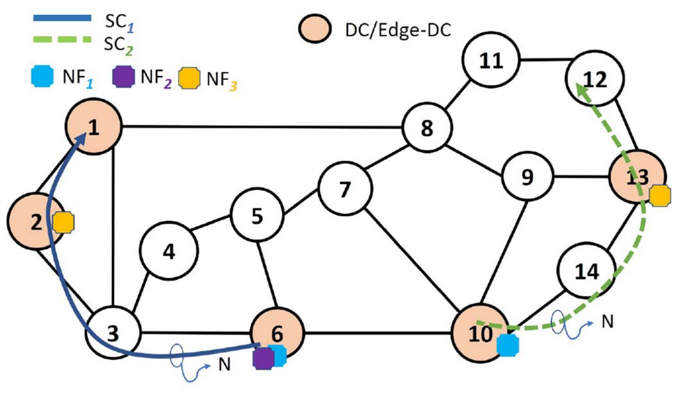


Fig. 1. Example deployment of service chains.

for the resources used (via *pay-per-use* payment methods). However, auto-scaling in cloud computing and auto-scaling SCs have some key differences: i) Cloud computing load vs. network traffic load: In cloud computing, workload mostly consists of computation tasks serving applications and users. On the other hand, SC workload consists of network traffic that requires to be served by network functions (NFs) (e.g., Firewalls, etc.). Due to the different types of workload (e.g., cloud VM hosting 'Pokemon Go' application server vs. service chain VM hosting 'Firewall VNF'), methods and results from cloud-computing domain are not directly applicable to SCs. Hence, we need to study novel auto-scaling methods which use network traffic load data (to learn from the network-specific load characteristics) and obtain new results demonstrating the impact of SC auto-scaling (instead of assuming that SC auto-scaling will have the same impact as cloud auto-scaling on cost, etc.) ii) Cloud QoS requirements vs. SC QoS requirements: Cloud services served by DC operators and service chains deployed by NOs usually have different QoS requirements. SC auto-scaling methods have to consider these different QoS requirements and deploy resources accordingly. Indeed, our proposed negotiation-game-based approach studies the impact of different levels of QoS sensitivity from tenants, and investigates its impact on cost savings, resource utilization, etc., which is a major novelty of this study. iii) Cloud resources vs. SC resources: In cloud computing studies, mostly virtual machines (VMs) (consuming compute and memory resources) are of focus. Hence, the cloud computing auto-scaling methods do not consider network bandwidth as a resource to be auto-scaled. On the other hand, for service chains, network bandwidth is critical, in addition to computing and memory resources consumed by NFs. Hence, novel SC auto-scaling methods [21]–[24] are being proposed to jointly auto-scale network and cloud resources.

Auto-scaling of SCs brings the economic benefit of *pay-per-use* for NO and tenants. The tenant pays lower leasing cost (due to lower resource usage) when the traffic load is low and higher leasing cost (due to higher resource usage) when the traffic load is higher. Tenants can reduce their leasing cost significantly by leasing network services from the NO which offers auto-scaled services. On the other hand, NO can attract more tenants by offering economic benefits of auto-scaling. NO can use the freed resources to serve SCs experiencing higher load and can even serve additional SCs (and additional tenants).

During the auto-scaling process, there can be a small duration of time, when the resources allocated to the tenant is not sufficient for the traffic load, leading to degraded service, negatively impacting QoS. Now, some tenants could tolerate a short duration of QoS degradation (e.g., video streaming). These tenants may be willing to auto-scale even when the economic gain is relatively low. But some other tenants could be very sensitive to QoS degradation (e.g., bank, stockbrokers, etc.) These tenants might participate in auto-scaling only when the economic gain is high.

Prior studies on SC have mostly focused on static and dynamic methods for SC resource placement [3], [6], [20], [22]. A few recent studies [21]–[24] have proposed heuristic algorithms to auto-scale SCs using prediction methods such as auto-regression [23]. However, these studies consider NO-driven auto-scaling, i.e., the NO allocates/de-allocates resources from SCs by assuming that tenants are always willing to release resources (or to accept additional resources), without considering the tenants' willingness to cooperate in the auto-scaling process, QoS sensitivity, etc.

In practice, some tenants may have more QoS-sensitive SCs (e.g., a bank requiring connectivity for financial operations) than others (e.g., a video content provider). Hence, depending on the services supported over the chain, the tenant may have preferences (e.g., willingness or unwillingness) to participate in auto-scaling. From an economic point of view, in this study, we assume that tenant's decision will also depend on how much refund it will gain by giving back resources to the NO.

In NO-driven auto-scaling method, the NO pays full refund of the tenant's leasing cost. But considering the fundamental economics, NO will be interested to pay less refund. Hence, NO's goal is to maximize the amount of re-collected free resources while paying minimum refund. To the best of our knowledge, our study is the first to explore a negotiation-game-based SC auto-scaling where tenants and NOs jointly participate in the auto-scaling decision based on their benefits.

Prior studies on auto-scaling (applied to SCs, or more generally, to cloud systems) have used *reactive* threshold-based approaches as well as *proactive* prediction methods (based on auto-regression, moving average, etc.) In this study, we enhance our previous ML prediction model in [8] and investigate an improved *proactive* Machine Learning (ML) based prediction methods (with different input, output, and feature set than those in [8]) to predict traffic demand for a given interval, ahead of time. We use the data collected from a private ISP to compare the performance of our proposed ML prediction method with the prediction methods proposed in prior studies.

We observe that each network function uses different number of computing and memory resources [20], depending on the traffic it is serving. But, in prior studies, these two resources are considered together as a single VM unit. In addition, many works ([18]–[19]) do not consider auto-scaling of network bandwidth at all. Hence, our proposed method models the problem with computing, memory, and network bandwidth as separate, dis-aggregated, and scalable resources.

Significant contributions of our study on the auto-scaling method are as follows:

1) To the best of our knowledge, we propose the first SC-negotiation-game-based auto-scaling method (Negotiated Resource Auto-Scaling for Service Chains (NRASC) algorithm) that allows the NO and tenants to engage in negotiation, based on QoS sensitivity and financial gain.
2) We propose another heuristic algorithm (Operator-driven Auto-Scaling for Service Chains (ORASC)) which allows us to compare the effectiveness of NRASC and the impact of our proposed prediction methods.
3) Our study explores various ML-based prediction methods including Deep Neural Networks (DNN) and Long Short-Term Memory (LSTM).
4) Our study compares our proposed method with three prior works: dynamic-threshold-based approach derived from cloud auto-scaling (DT) [10], a recent SC auto-scaling study (AR) [23], and an artificial-neural-network-based method [35]. Results demonstrate higher gain from our proposed methods.
5) We propose a SC resource model that considers disaggregation of resources (compute, memory, bandwidth, etc.).

The rest of this study is organized as follows. Section II reviews prior work on auto-scaling for cloud computing, VNFs, and SCs. Section III provides a formal problem statement for SC auto-scaling. Section IV describes the proposed ML-based prediction methods and two auto-scaling algorithms. Section V discusses the performance of ML-based prediction and shows numerical results on the auto-scaling algorithms. Section VI concludes the study.

II. Background and Related Work

A. *Auto-Scaling for Cloud Services*

Prior studies on auto-scaling for cloud computing can be classified in two groups: threshold-based vs. prediction-based (time series analysis, auto-regression, etc.) Threshold-based approaches have been used by DC owners [7] for scaling computing resources. Static-threshold-based approaches [9], [10], [11] use predefined upper and lower thresholds for scaling, which is not practical in a dynamic demand scenario. Improvements have been proposed using dynamic-threshold-based approaches [12], [13], [14]. As for prediction-based approaches, prior studies have used Auto-Regression (AR) [15], Moving Average (MA) [16], and Auto-Regressive Moving Average (ARMA) [17] to predict future workload for cloud virtual machine auto-scaling.

We have studied the proposed methods in cloud computing, adopted their threshold-based approach (DT), modified it for auto-scaling SCs, and we compare it with our proposed method in results section, to reflect the improvement over the *state-of-the-art*. In addition, a recent study [23] proposes a dynamic SC scaling method using auto-regression (AR), another method used in cloud auto-scaling. In results section, we compare our method with both DT and AR, and present improvements achieved by our proposed method.

B. *Auto-Scaling for Service Chains*

Prior studies on SC focus mostly on static and dynamic deployment of service chains. Reference [4], [6], [20] discuss the SC placement problem and propose mathematical optimization solutions for SC placement. Recent studies have started exploring SC auto-scaling. Reference [23] proposes a dynamic scaling method for a specific usecase: mobile core networks and IP Multimedia Subsystems (IMSs). The method uses auto regression (AR) as a prediction mechanism (we will compare it to our proposed ML method). Reference [24] proposed dynamic placement of VNF SCs across geo-distributed DCs. It converts the offline deployment problem into Online Regularization-Based Fractional Algorithm (ORFA).

Reference [25] proposes a heuristic VNF migration algorithm to reduce migration time and VNF embedding cost for SCs. The study focuses mostly on the migration decision algorithm, not on auto-scaling. Reference [26] proposes a heuristic algorithm for end-to-end latency-aware dynamic SC auto-scaling. Authors do not use any prediction method, assuming the workload is known from a Wikipedia trace data.

To summarize, references [23]–[26] do not distinguish among the SC resources (compute, network, storage, etc.), use traditional prediction methods (AR, ARMA, etc.), and do not consider tenant's participation in the auto-scaling decision.

As discussed, prior studies only consider NO-driven auto-scaling methods. To the best of our knowledge, our study is the first to explore the negotiation-game-based auto-scaling where tenants and NO decide on the auto-scaling based on their benefits (more in Section V). Another novelty of our work is the integration of ML-based prediction for auto-scaling.

In addition, our method is aware of disaggregated view of computing, memory, and bandwidth resources (instead of considering VMs as resource unit). Such disaggregation [27] has been explored in network resource literature. Using vertical scaling (adding/removing resources to/from a VM), disaggregation allows granular auto-scaling of resources.

III. Problem Description

Our objective is to develop auto-scaling methods which use the predicted values from machine learning classifier (see Fig. 2.a) to dynamically scale the SCs.

Fig. 2.a shows the overall flow of the auto-scaling process. The "Prediction" module learns from historic data and uses recent measurement data to predict the traffic demand during the next interval. Then, the "Scaling algorithm" makes necessary resource allocation/de-allocation based on the prediction.

The first step is to design a method to predict the amount of required resources in the next interval (φ). Section IV presents an enhanced version of the ML classifier proposed in [8]. Section V presents the SC auto-scaling algorithms. First, we propose NO-driven auto-scaling method, *ORASC algorithm*, which takes the output from *prediction* method and handles the complex SC resource management steps

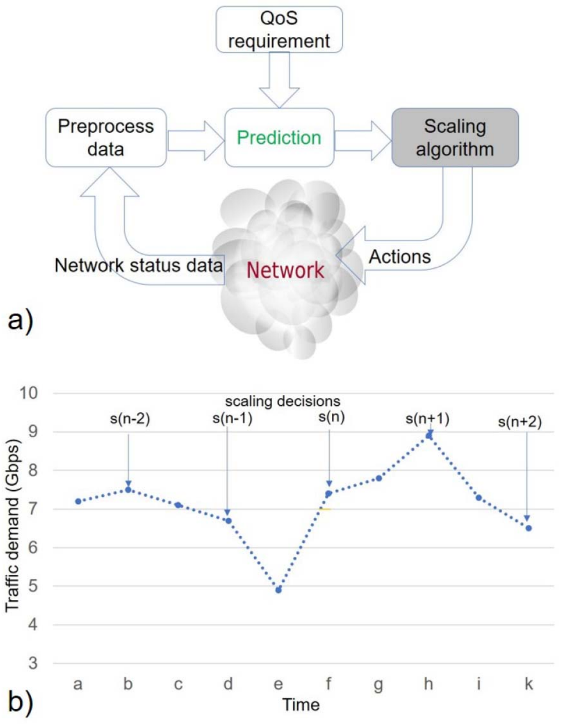


Fig. 2. Auto-Scaling overview: (a) high-level view of auto-scaling decision life cycle; (b) example steps for prediction decisions.

(such as allocating/de-allocating network capacity in links, adding/removing computing/memory units to VNFs, migration of virtual resource units, etc.) Then, we propose another auto-scaling algorithm based on negotiation game theory, *NRASC algorithm*, where tenants actively participate in the auto-scaling decision process. Our study envisions that, in future, the proposed method will be part of the automation, orchestration, and management software of the NO (similar to AT&T's ECOMP [28]).

Hence, the problem can be formally defined as follows: given a network topology, capacity of links, a set of nodes hosting compute and memory resources and their capacity limits, a set of SCs where tenant traffic flows are already assigned to a SC (using SC placement methods), minimum and maximum limits for each type of resources for each SC, historic measurement data to train ML algorithms, our method auto-scales SC resources, using the predicted *resource requirement* for the next interval. We assume that SCs do not share computing/memory/network units among each other, and traffic flows from one SC do not use another SC. In our study, we consider QoS is maintained if the required number of resources are allocated for that time period. On the other hand, if less resources are allocated than required, we consider QoS is violated, resulting in degraded QoS and QoS penalty.

The input parameters of the problem are as follows.

- Network topology $G(V, E)$: where V is set of network nodes with computing/memory hosting capability (DC or Edge-DC) and E is set of network links connecting the nodes in V.
- $e \,\epsilon\, E$: e is a link where e^t, e^u, and $(e^t - e^u)$ represent total, used, and available bandwidth (capacity) of link e, respectively.
- $v \,\epsilon\, V$: where v_c^t, v_c^u, and $(v_c^t - v_c^u)$ represent total, used, and available capacity of the CPU resource hosting node v, respectively. Also, v_m^t, v_m^u, and $(v_m^t - v_m^u)$ represent total, used, and available capacity of the memory resource hosting node v, respectively.
- *SC:* a set of service chains from different tenants (i.e., $SC = SC_1, SC_2, \ldots$).
- F: a set of traffic flows to be served by the SC (i.e., $F = F_1, F_2, \ldots$).
- NF_l: Virtual Network Function with type NF_l, indexed by l, hosting location given as NF_l^h, number of units deployed given as NF_l^χ.
- C: computing requirement in number of CPU cores.
- M: memory requirement in unit of GB.
- N: network capacity requirement (Gbps).
- $SC_q \,\epsilon\, SC$ represents qth SC. SC_q is defined by source (SC_q^s), destination (SC_q^d), assigned traffic flows ($F_q \subseteq F$), one or more VNFs ((NF_l)) hosting different VNF types, end-to-end network capacity requirement (N), route SC_q^r (traversing the VNFs) in which SC_q is allocated N capacity, VNF-specific compute (C) and memory (M) requirements for each VNF, and sequence is which VNFs are placed.
- Set of historic traffic load measurement data ($H(SC_q, t)$): indicates aggregated traffic (from the assigned traffic flows F_q) served by service chain SC_q at time t.
- For each SC deployment SC_q, upper and lower limits for the required resources are defined. For each C, M, and N of each NF_l inside SC_q, there is a minimum number of allowed units (e.g., C^{min}, M^{min}, N^{min}, etc.) and a maximum number of allowed units (e.g., C^{max}, M^{max}, N^{max}, etc).

IV. Proposed ML Classifier

In machine learning, an *instance* is a set of features/values representing a specific occurrence of the problem. For example, in our study, one *feature* of the problem instance is *time of day*. Another *feature* is value of the *measured traffic load* at a time of the day. We associate each instance (set of *features*) of the problem to a *class*, i.e., a classification decision. We convert the auto-scaling problem to a classification problem by training the classifier with a set of correctly-identified instances, called *training set*. In training phase, ML classifier learns a mapping between *features* and *classes*. After training phase, a classifier can be tested using a set of instances, called *test set*, which is not part of *training set*.

The Time vs. "Traffic demand" graph in Fig. 2.b is used to explain the input and output of the ML classifier. Fig. 2.b shows different timestamps (*a, b, c,* etc.) and auto-scaling decisions ($s(n-1)$, $s(n)$, $s(n+1)$, etc.), where $s(n)$ indicates the auto-scaling decision for the n-th interval. Let *measured network traffic load* for given timestamp x be $\lambda(x)$, and timestamp of auto-scaling step n be given by $\tau(n)$, where $\tau(n) - \tau(n-1) = \varphi$.

A. Feature Selection (Input)

Feature selection is the first important step towards defining the ML classifier. All our *features* are of numeric value. Referring to Fig. 2.a, we convert the traffic load measurements into the following features:

- Timestamp of decision: $\tau(n)$
- Day of Month (DoM): for $\tau(n)$
- Day of Week (DoW): for $\tau(n)$
- Weekday or Weekend (W): for $\tau(n)$
- Hour of Day (HoD): for $\tau(n)$
- Minute of Hour (MoH): for $\tau(n)$
- Measured traffic at time $\tau(n)$: $\lambda(\tau(n))$
- Traffic change from $\tau(n–1)$ to $\tau(n)$: $\lambda(\tau(n)) - \lambda(\tau(n-1))$
- Measured traffic at time $\tau(n-1)$: $\lambda(\tau(n-1))$
- Traffic change from $\tau(n–2)$ to $\tau(n-1)$: $\lambda(\tau(n-1)) - \lambda(\tau(n-2))$
- Measured traffic at time $\tau(n-2)$: $\lambda(\tau(n-2)) \ldots$
- Traffic change from $\tau(n-11)$ to $\tau(n-10)$: $\lambda(\tau(n-10)) - \lambda(\tau(n-11))$
- Measured traffic at time $\tau(n-11)$: $\lambda(\tau(n-11))$

We consider measured traffic up to n–11, giving us total 27 features, containing temporal information of measured traffic load and traffic load change from recent past. In Section VI, we explain how and why we choose these 27 features. Features 1-6 capture the temporal properties in the data, and rest of the features capture measured loads and how loads change over time. We explain the impact of these features on ML classifier using attribute selection algorithms, Principal Component Analysis (PCA), etc., in Section VI.

B. Class Definition (Output)

Next step is to define the output of the ML classifier, i.e., set of target *classes* that the classifier tries to predict. Note that, to provide a comprehensive prediction tool, we perform prediction of all resources (bandwidth, memory, and computing). But, as a certain amount of traffic requires a certain and pre-calculable amount of computing and memory capacity (depending on NF type, traffic type, etc.), we decided to derive conversion function from network bandwidth requirement to computing and memory requirement. Later, we present the conversion as functions; and, in Section VI, we show the numerical values that are used for conversion.

In our study, *class* depicts bandwidth requirement in Gbps, which is an integer value between N_k^{min} and N_k^{max}. To generate the labeled training set (*instances* with known *class* labels), we ensure that the scaling decision taken at step n allocates enough bandwidth to serve the traffic until the next decision-making step $n+1$.

To illustrate the trade-off between minimizing QoS violation and maximizing cost reduction, we propose two different approaches to generate the *class* values:

1) QoS Priority ML (QPML): In SC auto-scaling, there is a trade-off between QoS and cost reduction. We need to allocate more resources to guarantee QoS, but allocating more resources reduces cost saving. QPML gives priority to QoS over cost saving. To guarantee QoS, auto-scaling decision at step n (present) considers traffic changes until the next auto-scaling step $n + 1$. QPML generates the *class* value as follows:

$$s(n) = max(\lambda(t)), \forall t\epsilon\{\tau(n), \ldots, \tau(n+1)\} \quad (1)$$

where t is timestamp with traffic data between steps n and $n + 1$ (including $\tau(n)$ and $\tau(n + 1)$).

2) Cost Priority ML (CPML): In some cases, network owner/leaser may choose to ignore short-lived bursty traffic between steps n and $n + 1$ to save cost by avoiding over-provisioning of resources and accepting short-lived degradation. CPML considers measured traffic load only at step n (present) and at next auto-scaling step $n + 1$ (future). CPML generates the *class* value as follows:

$$s(n) = max(\lambda(\tau(n)), \lambda(\tau(n+1))) \quad (2)$$

where $\tau(n)$ is the time at which step i takes place and $\tau(n + 1)$ is the time when step $n + 1$ occurs.

TABLE I
EXAMPLE INSTANCES WITH KNOWN LABELS

Case	Fea. 1	Fea. 2	Fea. 3	Fea. 4	...	QPML Class	CPML Class
1	15	3	1	2		$qos(\lambda(h))$	$qos(\lambda(\tau(n)))$
2	15	3	1	2		$qos(\lambda(h))$	$qos(\lambda(\tau(n+1)))$

C. Data Generation

After defining the input and output of the classifier, the next task is dataset generation. For training-set and test-set generation, we assume that realistic traffic-load measurement data $H(SC_q, t)$ is available for each of the SCs. Table I shows an example of training instance for QPML and CPML for scaling decision at steps n–1 and n (Fig. 2) where *f,g,h,i,* etc. are time values.

D. Machine Learning Algorithms

Selecting the right ML algorithm is the next task towards training the classifier. We explore different algorithms in the ML suite WEKA [30], including decision-tree-based algorithms (Random Tree, J48, REPTree, Random Forest), rule-based algorithms (Decision Table), artificial neural networks (ANN), and Bayesian-network-based algorithms (BayesNet). A brief introduction to the algorithms is covered in [8].

In our study, we also compare the accuracy of the machine learning algorithms with Deep Neural Networks (DNN) and Long Short-Term Memory (LSTM). DNN [34] is an artificial neural network (ANN) with multiple layers between the input and output layers. DNN can extract complex relationship between the input and the output. This has made DNN a successful prediction method in many studies.

LSTM [33] is an artificial recurrent neural network (RNN) architecture often used in the field of deep learning. We have used 'ADAM algorithm', an adaptive learning rate optimization algorithm. In contrast to DNN, LSTM has loops, allowing prediction of future events while remembering past events. Detail architecture and design of DNN and LSTM can be found in [34] and [33], respectively.

E. Performance Evaluation

A test dataset is used to evaluate the performance of the trained classifier. Given a trained classifier and a test set, the test outcome is divided into four groups: i) True Positive (TP): positive instances correctly classified; ii) False Positive (FP): negative samples incorrectly classified as positive; iii) True Negative (TN): negative samples correctly classified; iv) False Negative (FN): positive samples wrongly classified as negative.

We consider three different performance metrics:

(a) Precision (%): Precision corresponds to the fraction of predicted positives which are in fact positive. Precision is a strong indication of accuracy for the classifier. Precision is given by percentage of: *TP*/(*TP* + *FP*).

(b) False Positive (%): FP is an important indicator of classifier as lower FP indicates less classification mistakes.

(c) ROC area: Receiver Operating Characteristic (ROC) curve is a graphical plot in which true positive rate (*TP*/((*TP* + *FN*)) is plotted as function of the false positive rate (*FP*/(*FP* + *TN*)). ROC area is a robust metric for classifier performance.

V. SC Auto-Scaling Methods

In this section, we propose two SC auto-scaling methods: (i) Operator-driven and (ii) Negotiation-game-driven. The Operator-driven SC scaling algorithm (ORASC) helps us to quantify and compare different prediction methods from the literature with our proposed machine learning approach (Figs. 7-9 in results). To mimic traditional SC scaling methods, ORASC considers fixed level of QoS sensitivity and fixed refund. On the other hand, negotiation-game-driven SC scaling algorithm (NRASC) is our novel SC scaling method, which is the first to propose a solution for multiple levels of QoS sensitivity (for tenants) and negotiated refund (between tenants and NO). Later, we compare ORASC and NRASC to demonstrate the win-win scenario brought by NRASC (see Figs. 10-13 in results).

Both methods utilize prediction methods to auto-scale SC resources. The methods consider Network Management and Orchestration (NMO) and Distributed Cloud Management (DCM) entities which control network and compute/memory resources, respectively. Also, prediction methods are re-trained after a certain interval to keep them up to date. Re-training threshold σ determines if prediction methods need to be re-trained.

A. Operator-Driven SC Scaling Method (ORASC)

In Operator-driven auto-scaling (Algorithm 1), NO uses predicted demand to allocate/de-allocate resources from SCs assuming that tenants are always willing to release resources (or to accept additional resources).

Algorithm 1 uses the predicted bandwidth requirement (line 4) to determine the required number of computing (line 6)

Algorithm 1 Operator-Driven Resource Auto-Scaling for Service Chains (ORASC)

```
1:  Initialize with given input parameters in Section III;
2:  Train prediction methods (Section V) for each $SC_i$ using
    input data $H(SC_q, t)$;
3:  for each $SC_q$ in SC do
4:      $N^p \leftarrow$ prediction method output for $SC_q$;
5:      for each $NF_l$ served in $SC_q$ do
6:          $C^p \leftarrow f1(N^p, NF_l)$;      ▷ amount of computing units
7:          $M^p \leftarrow f2(N^p, NF_l)$;  ▷ amount of memory units
8:          $\zeta \leftarrow true$;
9:          RAM($C, C^p, M, M^p, N, N^p, NF_l, SC_q, \zeta$);
10:     end for
11: end for
12: if $\sigma$ is expired then
13:     go to line 2;
14: else
15:     Update $\sigma$;
16:     go to line 3;
17: end if
```

and memory (line 7) resources for each NF, using bandwidth-to-computing conversion function $f1(.)$ and bandwidth-to-memory conversion function $f2(.)$, respectively. In line 9, the algorithm calls function RAM(.) (i.e., Algorithm 2) with appropriate parameters. RAM(.) scales up (lines 6-34) or down (lines 36-42) the resources according to the demand change. RAM(.) also handles scenarios such as not enough resources at the DC/Edge-DC (lines 23-33), resource requirement did not change (lines 3-4), and tenant demand has grown so much that QoS can not be satisfied with the current SC resources; so, NO needs to deploy an additional SC in a new route where we have enough (computing, storage, and bandwidth) resources to serve the tenant (lines 44-49).

Run-time complexity of ORASC is *O*(*q***l*), where q = number of SCs and l = average number of NFs in SCs.

B. Negotiation-Game-Driven SC Scaling Method (NRASC)

ORASC proposes the traditional way of auto-scaling chains where the operator takes the auto-scaling decision based on performance metrics, without consulting tenants' preferences. But, in practice, depending on the usecase of the service chain, the tenant may have preferences (e.g., willingness or unwillingness) to participate in auto-scaling.

From QoS point of view, tenant's decision will depend on the priority level of the service chain. For example, a bank handling sensitive information might be more risk-averse than a video-streaming service. Hence, a bank will be less willing to auto-scale its SCs (keeping all free resources to itself), compared to the video service provider.

From economic point of view, tenant's decision will depend on how much refund it will gain by giving back free resources to NO. In ORASC, we assume the NO pays full refund to tenants. But, in practice, NO will be interested to pay less refund. Hence, NO's goal is to maximize free resource collection while paying minimum refund. Also, tenants with higher

Algorithm 2 Resource Allocation Method (RAM)

```
1: Input: $C, C^p, M, M^p, NF_l, SC_q, \zeta$;
2:                                        ▷ unchanged requirement
3: if $C^p == C$ & $M^p == M$ then
4:     Load balance (DCM) traffic flows in $F_q$;
5:                                        ▷ more resources required
6: else if $C^p > C$ & $M^p > M$ then
7:     $v \leftarrow NF_l^h$;
8:     $\delta_i \leftarrow C^p - C$;
9:     $\delta_j \leftarrow M^p - M$;
10:    $\delta_k \leftarrow N^p - N$;
11:                                       ▷ enough resources at $v$
12:    if $(v_c^t - v_c^u) > \delta_i$ & $(v_m^t - v_m^u) > \delta_j$ then
13:        if $SC_q^r$ does not have $\delta_k$ additional bandwidth then
14:            Find new route $SC_q^{r'}$ with $N^p$ bandwidth;
15:            $SC_q^r \leftarrow SC_q^{r'}$;
16:            Re-route traffic through $SC_q^r$;
17:        end if
18:        Allocate $\delta_i$ additional $C$s and $\delta_j$ additional $M$s,
19:        to $NF_l$ at location $v$;
20:        Load balance (DCM) traffic flows in $F_q$;
21:                                       ▷ not enough resources at $v$
22:    else
23:        $v' \leftarrow$ DCM finds optimum location to host $NF_l$;
24:        Allocate $C^p$ and $M^p$ at $v'$ via DCM;
25:        Migrate necessary data from $NF_l$ instance at $v$
26:        to $v'$ via DCM;
27:        Turn on $NF_l$ instance at $v'$ with allocated compute
28:        and memory resources and migrated data;
29:        Find new route $SC_q^{r'}$ with $N^p$ bandwidth
30:        through $v'$;
31:        $SC_q^r \leftarrow SC_q^{r'}$;
32:        Re-route traffic through $SC_q^r$;
33:        Release $C$ & $M$ at $v$ via DCM;
34:    end if
35:                                       ▷ remove extra resources
36: else if $C^p < C$ & $M^p < M$ & $\zeta == true$ then
37:    $v \leftarrow NF_l^h$;
38:    $\delta_i \leftarrow C - C^p$;
39:    $\delta_j \leftarrow M - M^p$;
40:    DCM finds optimum $\delta_i C$s and $\delta_j M$s to de-allocate
41:    from $NF_l$ instance at $v$ location;
42:    Load balance traffic flows in $F_q$;
43:                                       ▷ SC QoS upper limit
44: else if $C^p > C^{max}$ then
45:    Consult NMS, DCM, and placement algorithm to
46:    create a new SCs, $SC_q'$;
47:    Reroute and load balance flows from
48:    $F_q$ between $SC_q$ and $SC_q'$;
49: end if
```

QoS requirement will release free resources only if the NO is paying high refunds. Our proposed negotiation-game-based method allows the NO and its tenants to negotiate such that both can achieve a win-win equilibrium.

Let the list of unused resources in service chain SC_q for next interval σ be given by SC_q^X. Leasing cost paid by tenant for current unused (and available resources) in SC_q is expressed as SC_q^L. NO's 'Offered Refund (OR)' for a certain negotiation round is expressed by R and 'Refund Ratio', α, is the ratio between 'Offered Refund' and leasing cost of unused resources, $\alpha = R/SC_q^L$.

Tenant's 'Expected Refund (ER)' for service chain SC_q is expressed as SC_q^α. SC_q^α denotes that tenant is willing to participate in auto-scaling (i.e., releasing its unused resources) if $\alpha > SC_q^\alpha$.

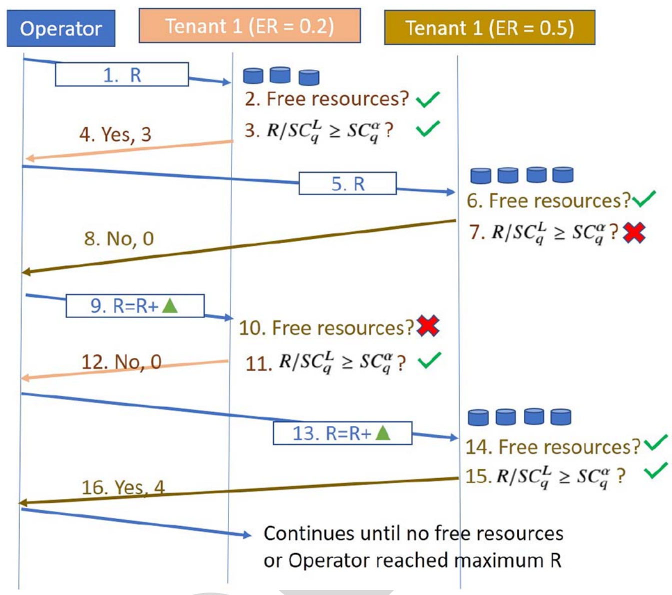

Operator
Tenant 1 (ER = 0.2)
Tenant 1 (ER = 0.5)
1. R
2. Free resources? ✓
3. $R/SC_q^L \geq SC_q^\alpha$? ✓
4. Yes, 3
5. R
6. Free resources? ✓
7. $R/SC_q^L \geq SC_q^\alpha$? ✗
8. No, 0
9. R=R+▲
10. Free resources? ✗
11. $R/SC_q^L \geq SC_q^\alpha$? ✓
12. No, 0
13. R=R+▲
14. Free resources? ✓
15. $R/SC_q^L \geq SC_q^\alpha$? ✓
16. Yes, 4
Continues until no free resources or Operator reached maximum R

Fig. 3. Example negotiation between NO and tenants.

Fig. 3 explains the negotiation game between NO and tenants with an example. At step 1, NO offers refund *R* to Tenant 1 (Expected Refund (ER) = 0.2). As refund *R* satisfies the tenant's expected refund level, the tenant agrees to auto-scale, releasing 3 free resources. But, for Tenant 2 (ER = 0.5), refund *R* is not enough, hence it rejects to participate in auto-scaling.

In the next round, NO increases the 'Offered Refund, R' with Δ amount and re-starts negotiating. Now, R is sufficient for Tenant 2, so it agrees to participate in auto-scaling. These negotiation rounds keep going on until there are no free resources or NO has reached maximum R (equilibrium).

Fig. 3 shows the example of the negotiation game showing a few rounds of negotiation going on between the NO and the tenants in runtime. However, the implementation in NRASC (Algorithm 3) does not require physical communication between the tenants and NO. In NRASC algorithm, the tenants' parameters required for the negotiation rounds are available at the beginning of the negotiation game. NO executes the NRASC by looking into the parameters (e.g., expected refund, etc.) already defined by the contracts with the tenants.

Algorithm 3 uses similar negotiation steps in the context of service chains. Lines 3-19 create a candidate list for auto-scaling with SCs which have free resources. Line 21 starts with an initial R and starts the negotiation with SCs from the candidate list. If SC agrees (line 25) to auto-scale for the given R, the algorithm calls RAM(.) functions for each NF with appropriate parameters. After one round of negotiation,

Algorithm 3 Negotiated Resource Auto-Scaling for Service Chains (NRASC)

```
1:  Initialize with given input parameters in Section III;
2:  Train prediction methods (Section V) for each $SC_i$ using input data $H(SC_q, t)$;
3:  candidate_list ← empty
4:  for each $SC_q$ in SC do
5:      $N_k^p \leftarrow$ prediction method output for $SC_q$;
6:      for each $NF_l$ served in $SC_q$ do
7:          $C^p \leftarrow f1(N^p, NF_l)$;
8:          $M^p \leftarrow f2(N^p, NF_l)$;
9:      end for
10:     Calculate $SC_q^{\chi}$;
11:     if $SC_q^{\chi}$ is not empty then                ▷ unused resources
12:         candidate_list ← candidate_list ∪ $SC_q$
13:     else                                   ▷ may need additional resources
14:         for each $NF_l$ served in $SC_q$ do
15:             $\zeta \leftarrow false$;
16:             RAM($C, C^p, M, M^p, N, N^p, NF_l, SC_q, \zeta$);
17:         end for
18:     end if
19: end for
20:                                                ▷ Negotiation starts
21: Initialize R
22: while (unused resources available || NO has not reached maximum R) do
23:     for each $SC_q$ in candidate_list do
24:         NO offers R to $SC_q$ Tenant;
25:         if $R/SC_q^L \geq SC_q^{\alpha}$ then
26:                                            ▷ Tenant agrees to release
27:             $\zeta \leftarrow true$
28:             for each $NF_l$ served in $SC_q$ do
29:                 RAM($C, C^p, M, M^p, N, N^p, NF_l$,
30:                 $SC_q, \zeta$);
31:             end for
32:         end if
33:         NO moves to next Tenant with same offer R;
34:     end for
35:                                       ▷ Offer more refund in next round
36:     $R \leftarrow R + \Delta$
37: end while
```

line 37 updates R. This process keeps going on until there are no free resources or NO has reached maximum R. Runtime complexity of NRASC is $O(q*l + x*q*l)$, where q = number of SCs, l = average number of NFs in SCs, and x = average number of negotiation rounds.

We are aware that, in special circumstances, tenants can be tempted to engage in collusion (e.g., share information with one another) to maximize their refund, leading to a broken negotiation process. We have explored existing mechanisms to avoid collusion between tenants, and we have identified a vast literature from both technical [36], [37] and legal [38] domains. SC scaling is much different than the traditional game theory problems where we have multiple bandits/outlaws who do not obey the law. The tenant and the NO are bound by the legal and technical contracts which limit the tenants ability to share sensitive information. Breaking the legal contracts will lead to huge financial loss for the tenant compared to the gain achieved by the collusion. Hence, we can safely illustrate that NOs (or any other organization handling hundreds of competing tenants over many years) have mechanism in place to avoid collusion between tenants. Scope of our study is not sufficient to discuss and cover all these mechanisms. However, we assume that NO will consider using technical [36], [37] and legal [38] mechanisms to protect the negotiation process from collusion.

C. Leasing Cost Model

We propose this leasing cost model to show the cost savings from the tenant's point of view. Let us define:

L_t: Total leasing cost tenant pays for the service.

L_C: Leasing cost per unit (CPU core) of compute resource per hour.

L_M: Leasing cost per unit (GB) of memory per second.

L_N: Leasing cost per unit (Gbps) of bandwidth per hour.

L_P: Penalty per hour due to degraded service (under-provisioning). Revenue that tenant loses if the service does not maintain QoS.

β_q: Duration of service for SC_q.

γ_q: Duration of degraded service for SC_q.

Leasing cost for one VNF instance NF_l:

$$NF_l^L = (C * L_C + M * L_M + N * L_N) * \beta_q \tag{3}$$

During a SC's life time, auto-scaling methods may introduce QoS degradation due to under-provisioning of resources (e.g., 7 Gbps bandwidth is required, but the forecast was 6 Gbps). Our study captures this degradation of service as a duration of degraded QoS. We model the revenue penalty occurred by this duration spent in degraded QoS using Eqn. (4). Leasing cost for one SC SC_q, including QoS degradation penalty, is:

$$SC_q^L = \sum_l \left(NF_l^L\right) + L_P * \gamma_q. \tag{4}$$

VI. Illustrative Numerical Examples

This section provides performance analysis of the proposed ML classifier for prediction and then presents results for ORASC and NRASC.

A. ML Classifier Prediction Accuracy

1) Experimental Setup for ML Classifier: To generate "features" and "classes" for training and testing of the ML classifier, we use realistic traffic load traces from [35]. Traffic load data (in bits) was collected at every five-minute intervals over a 1.5-month period from a private ISP and on a trans-Atlantic link [39]. Maximum traffic load in the data is 10 Gbps, so we use 10 Gbps as maximum traffic processing at any SC deployment (N^{max}), and minimum N^{min} is considered as 1 Gbps. As traffic load traces are at every 5-minute interval, we use a prediction interval (φ) = 10 minutes. However, our methods are generic to work for other intervals (explained with Table IV).

TABLE II
PREDICTION ACCURACY OF THE PROPOSED CPML CLASSIFIER

ML algorithm	Precision (%)	False Positives (%)	ROC Area (%)
Random Forest	96.5	0.7	99.7
Random Tree	91.4	1.7	94.7
J48	94.9	1	98.5
REPTree	95.1	0.9	99.3
Decision Table	95.5	1	99.6
ANN	92.8	1.4	99.4
BayesNet	90.9	1.7	99.5
DNN	95.1	0.7	99.8
LSTM	95.7	0.7	-

TABLE III
PREDICTION ACCURACY OF THE PROPOSED QPML CLASSIFIER

ML algorithm	Precision (%)	False Positives (%)	ROC Area (%)
Random Forest	95.5	1.2	99.4
Random Tree	90.9	2	93.9
J48	93	1.6	97.3
REPTree	94.2	1.3	99.1
Decision Table	92.8	1.5	99.2
ANN	91.3	1.8	98.3
BayesNet	90.3	1.9	99.1
DNN	92.4	0.7	99.8
LSTM	93.1	1.6	-

ML settings used in WEKA [30] are as follows:

- Random Forest: batch size = 100, number of iterations = 100.
- Random Tree: batch size = 100, minimum variance proportion = 0.001.
- J48: batch size = 100, confidence factor = 0.25.
- REP Tree: batch size = 100, minimum variance proportion = 0.001.
- Decision Table: batch size = 100, search method: best first (hill-climbing algorithm with backtracking).
- Artificial Neural Networks (ANN): batch size = 100, learning rate = 0.3.
- BayesNet: batch size = 100, search algorithm = K2 (hill-climbing algorithm).
- DNN: batch size = 100, learning rate = 0.3.
- LSTM: optimization algorithm: ADAM algorithm.

2) Prediction Accuracy of Proposed ML Classifiers: Here, we consider the three performance metrics discussed in Section IV-E to explain the accuracy of the proposed methods. In Table II, we compare the prediction accuracy of the CPML prediction method for different ML algorithms. For results in Tables II and III, we use 40 days of data (40*144 = 5760 instances) for training and two days of data (2*144 = 288 instances) for testing. We assume that the ML model is retrained with new data every two days (σ). First, Tables II and III shows that, among different ML algorithms used to train the classifier, "Random Forest" has higher precision for both QPML (95.5%) and CPML (96.5%) approaches. Difference in prediction accuracy is due to different "class" generation results (from Eqn. (1)) than CPML (from Eqn. (2)). Note that, LSTM does not have the ROC Area (%) evaluation metric.

False Positives are important metric for ML classifiers. If a ML classifier generates too many false positives, in most scenarios, the classifier will not be considered as a recommended one. Tables II and III report False Positives (lower is better) for QPML and CPML with different ML algorithms. Again, "Random Forest" shows lowest FP with 1.2% for QPML and 0.7% for CPML.

ROC Area is another important and robust metric for ML prediction models. ROC Area considers the performance of ML classifier in complete range of true positives and false positives, and then reports overall performance of the classifier. Tables II and III show ROC Area (%) for QPML and CPML (higher is better). Again, Random Forest shows highest ROC Area with 99.4% for QPML and 99.7% for CPML.

Decision-tree-based algorithms perform better for the prediction accuracy compared to artificial-neural-network-based and Bayesian-network-based algorithms. Among decision-tree-based algorithms, "Random Forest" leads with highest precision (96.5%), highest ROC Area (99.7%), and lowest false positives (0.7%). Clearly, the pattern of the data and feature set favor decision-tree-based algorithms to learn, as the tree-based algorithm keeps all the intermediate tree classifiers during the learning process. In addition, "Random Forest" further improves the performance by averaging/voting between several tree-classifier instances. However, ANN-based model uses backpropagation by gradient descent to set the weights of neurons' connections. Such ANN-based models suffer from a fixed learning issue, where the model fails to learn new trends once the weights are fixed [41]. This limitation causes lower performance for ANN, compared to decision-tree-based method such as Random Forest.

TABLE IV
IMPACT OF PREDICTION INTERVAL (φ) VARIATION ON PRECISION (%)

ML Algorithm	5 minutes	10 minutes	15 minutes
Random Forest	95.3	96.5	93.8
Random Tree	95.3	91.4	90.4
J48	95.2	94.9	92.8
REPTree	93.9	95.1	92.8
Decision Table	94.0	95.5	93.3
ANN	94.2	92.8	90.4
BayesNet	95.4	90.9	87.1
DNN	94.4	95.1	91.6
LSTM	93.8	95.7	91.4

Table IV shows the impact of prediction interval (φ). We consider intervals of 5, 10, and 15 minutes for CPML. We observe that, for prediction interval of 5 minutes, Random Forest (95.3%), Random Tree (95.3%), and BayesNet (95.4%) present high accuracy. For prediction interval of 15 minutes, Random Forest (93.8%) performs with the highest accuracy. On the other hand, as we move away from $\varphi = 10$ minutes to $\varphi = 15$ minutes, the accuracy reduces slightly. Explanation of this phenomenon is that, for longer φ, the prediction accuracy depends on more data points, limiting the performance of ML model. But, it is safe to conclude that Random Forest is performing steadily for all three prediction intervals. We have explored similar results for QPML as well. Hence, for the rest of the numerical evaluations, we use results from "Random Forest".

3) Learning Curve Analysis (Impact of 'Number of Features' and 'Training Dataset Size' on Prediction Accuracy): Figs. 4-5 provide learning curve of the proposed

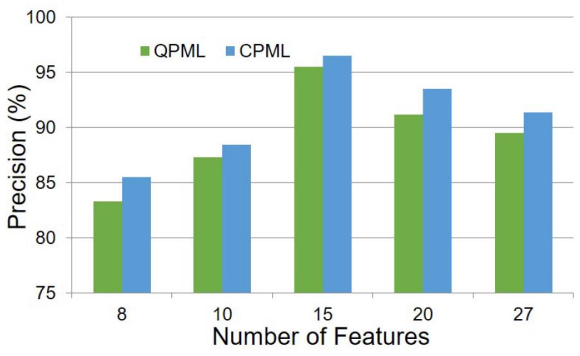


Fig. 4. Number of features vs. prediction accuracy in precision (%).

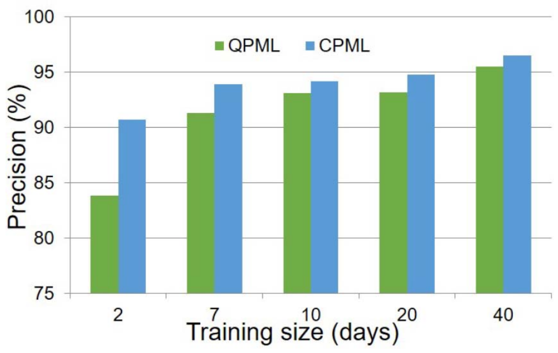


Fig. 5. Training data size vs. prediction accuracy in precision (%).

ML classifier. Learning-curve analysis presented here helps us to understand the following two important aspects of training a classifier: "How many features generate best results?" and "Does more data help or not?"

Our study did not start with fixed 27 features. Instead of deciding the number of features upfront, we explored different number of features and compared their performances, to determine the best performing configuration. We started by considering a small number of features (8), observed the highest accuracy with 15 features, and then increased up to a number (27), which clearly shows that the performance is not improving with more features. As shown in Fig. 4, 15 features have the highest accuracy, with accuracy going up starting from 8 features, and going down for 20 and 27 features. Hence, 27 features were the number of features we needed to report this complete picture. For example, number of features "8" means we are using only the first eight features from Section IV-A. As shown in Fig. 4, accuracy of ML classifier increases with number of features. But, after number of features exceeds "15", accuracy decreases. This means that, if we keep adding more features by moving away from the time of prediction, the additional *features* impact the accuracy negatively. Hence, we use 27 features only to decide the highest performing 'number of features'. Once decided, for the rest of the results (shown in Tables II-VI and Figs. 5-13), we use 15 features.

Fig. 5 shows impact of training dataset size on prediction accuracy (precision) of ML classifier. The general intuition is that, with more data, ML classifier should perform better. We observe that *7 days of training data* has significant performance improvement over *2 days of training data*. One explanation of this phenomenon is that *7 days of training data* offers insights from seasonal pattern and periodicity of the load during the whole week (compared to 2 days). Then, *10 days of training data* improves the ML model further, but *20 days of training data* does not have much additional learning points. Then, *40 days of training data* introduces the monthly pattern and improves the precision significantly more than 20 days. For rest of the study, we consider 15 features and 40 days of training data.

TABLE V
IMPACT OF DIFFERENT FEATURES

Measured Load	DoM	DoW	W	HoD	CPML
✓					96.1
✓	✓	✓	✓	✓	96.2
	✓	✓	✓	✓	83
✓	✓				95.8
✓			✓		95.5

4) More Insights (Feature Ranks and Impact of Different Features on Prediction Accuracy): Important questions regarding ML classification are: "Can we identify the dominant features from the feature list?" "Which features impact classification accuracy more?" "Can we explain how different combination of features impact the accuracy?" Below, we explore different methods to answer these questions.

First, we use attribute (feature) selection algorithm *InfoGainAttributeEval* [40] from WEKA which evaluates the importance of a feature by measuring the information gain with respect to classification. After ranking the first 15 features, features 7, 9, 11, 13, and 15 are ranked 1 through 5, in that order. This observation gives two important insights: a) "Measured loads" are most important features that contributed to accurate classification; and b) "Measured loads" closer to decision time have more significant impact on classification.

We have confirmed this observation using Principal Component Analysis (PCA), a statistical procedure (often used with feature-ranking methods) to find correlated features and their impact on classification. In PCA, the first principal component has the largest possible variance which accounts for much of the variability in the data. Our PCA reports a combination of features 7, 9, 11, 13, and 15 as the first principal component, conveying similar takeaway as *InfoGainAttributeEval*. Feature 2 (day of week), feature 1 (day of month), and feature 3 (weekday or weekend) are ranked 6th, 7th and 8th, respectively. As expected, these three features carry information related to the temporal variation of load, so they are ranked highly in feature ranks. Rest of the features are ranked in the following sequence: 14, 12, 6, 4, 8, 10, and 5.

Table V shows impact of different features on auto-scaling decision accuracy. We compare the precision of the algorithms with different combination of features such as "Measured Load" (features 7, 9, 11, 13, and 15), DoM (Day of Month), DoW (Day of week), W (Weekday or Weekend), HoD (Hour of Day), etc. As discussed earlier, only "Measured load" feature set shows very high precision. Then, in second row, "Measured Load" with rest of the temporal features improves accuracy to 96.2%. Only temporal features (row 3) show

TABLE VI
TRAINING AND TESTING TIMES OF ML CLASSIFIER

ML Algorithm	Training Time (ms)	Testing Time (ms)
Random Forest	1770	20
Random Tree	50	0
J48	280	20
REPTree	150	0
Decision Table	700	0
ANN	16690	0
BayesNet	170	10

significant accuracy of 83%. One interesting observation is that, if we pick a single temporal feature with "Measured Load" features, performance degrades. This means temporal features can help to improve decisions only when they work together to provide the complete seasonal variations and patterns.

5) Training and Testing Time of ML Classifiers: Our study assumes that the ML classifier will run in real-time to provide auto-scaling decisions. Also, the model will be retrained every two days with updated data. Hence, it is important to report the training (off-line model building) and testing (run-time decision making) times of the ML classifiers. Table VI shows training time (5760 instances, 40 days of data) and testing time (288 instances, two days of data) for different ML algorithms. WEKA reports times in seconds upto two digits after decimal point. This means the zeros reported in the table take milliseconds or less time to make 288 auto-scaling decisions, which is very promising for real-time deployment for our method. Also, training times are few seconds or less, which supports retraining the model every two days.

To compare the algorithms, "MLP" (neural network) takes longest (16.69s) and "Random Tree" takes shortest (0.05s) to train the models. In run-time, we see many sub-millisecond algorithms such as "Decision Table". On the contrary, "Random Forest" takes 0.02 seconds. This is an important decision-making point. For example, in a special case, if the NO is willing to accept slight loss of accuracy (Decision Table 95.6% vs. Random Tree 96.5%) to obtain faster decisions, "Decision Table" can be a better choice than "Random Forest". Such practical consideration related to ML-based solutions is a strong motivation for our study.

6) DNN and LSTM Performance Analysis: For DNN and LSTM, we have used the same dataset, feature set, and training and testing data ratio as in the previous part of the study.

As the number of hidden layers are a key parameter for the performance of the DNN, in Fig. 6 we report the accuracy of DNN model with respect to number of hidden layers. We observe that, with growing number of hidden layers, the accuracy of prediction increases (up to 95.1% for CPML). With number of hidden layers = 15, for both CPML and QPML, DNN (95.1% and 92.4%, respectively) performs better than ANN (92.8% and 91.3%, respectively). However, from hidden layers = 20, the accuracy starts decreasing. This phenomenon can be explained by the fact that, as DNN starts adding more and more hidden layers, it starts overfitting, thus resulting in lower accuracy.

We also observed the impact of prediction interval (φ) on prediction accuracy of DNN. Table VII reports the impact of three different decision intervals (φ): 5, 10, and 15 minutes. Again, in all three interval scenarios, DNN outperforms ANN: 5 minutes (DNN 94.4% vs. ANN 94.2%), 10 minutes (DNN 95.1 vs. ANN 92.8%), and 15 minutes (DNN 91.6 vs. ANN 90.4%).

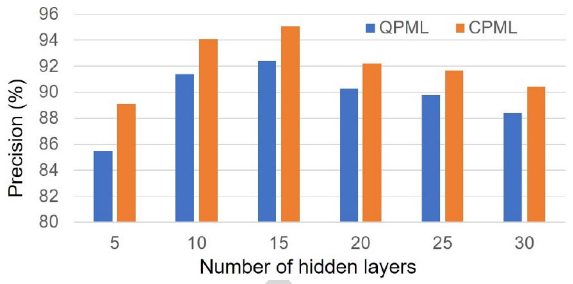


Fig. 6. Number of hidden layers vs. prediction accuracy in precision (%) for DNN.

TABLE VII
IMPACT OF PREDICTION INTERVAL (φ) VARIATION ON ACCURACY OF DNN

No. of hidden layers	5 minutes	10 minutes	15 minutes
5	93.0	89.0	89.4
10	93.7	94.1	89.5
15	94.4	95.1	91.6
20	93.7	92.2	89.9
25	92.6	91.7	89.6

We have also explored LSTM-based prediction using the same dataset. We have used Deeplearning4j [42] to implement the LSTM method. The LSTM-based method shows a promising accuracy of 95.7%, which is higher than DNN (95.1%) and closely comparable to the highest-performing ML algorithm Random Forest (96.5%). The reason LSTM can not outperform Random Forest can be explained from two aspects: i) LSTM is more suitable for time series data prediction [33], while, in our study, we have modeled the problem as a classification problem; and ii) long-term information inside the LSTM model can often get corrupted (discussed in detail in [43]), and data of our study is dependent on both short-term daily variations and long-term weekly and monthly variations.

B. Experimental Setup for Auto-Scaling of Service Chains (ORASC and NRASC)

Our study considers that service-chain requests arrive from different tenants, and NO allocates necessary resources. During the SC's life time, resource requirement varies and the auto-scaling algorithms allocate/de-allocate resources as necessary. For rest of the results, we use QPML as the ML algorithm. We use the NSFNet topology from Fig. 1 with DCs at nodes 2, 6, 7, and 13. Each link has 400 Gbps capacity. We consider each DC with 1000 racks capacity. We consider rest of the nodes as Edge-DCs with 5 racks capacity. In our study, each rack has 2 hypervisors and 4 servers with 8 core CPU each. We consider 2 GB memory for each core of CPU. We also consider the following: 1 CPU core leasing cost $L_C = 0.002$ dollar per hour [44], 1 GB memory leasing cost $L_M = 0.001$ dollar per hour [44], 1 Gbps bandwidth leasing cost $L_N = 0.0972$ dollar per hour [45].

TABLE VIII
SERVICE CHAIN TYPES AND PERCENTAGE IN SIMULATION

Service chain	Traffic (%)	VNF chain
Web Service	20	NAT-FW-TM-WOC-IDPS
VOIP	10	NAT-FW-TM-FW-NAT
Video Streaming	60	NAT-FW-TM-VOC-IDPS
Online Gaming	10	NAT-FW-VOC-WOC-IDPS

TABLE IX
CONVERSION TO COMPUTE (CPU) RESOURCES ($f1(.)$)

VNF	Bandwidth requirement (Gbps)									
	1	2	3	4	5	6	7	8	9	10
NAT	1	1	1	1	1	2	2	2	2	2
FW	1	1	2	2	4	4	8	8	16	16
TS	1	2	4	4	8	8	8	12	12	16
WOC	1	1	1	1	2	2	2	2	2	4
IDPS	1	1	2	2	4	4	8	8	16	16
VOC	2	2	4	4	8	8	8	12	12	16

TABLE X
CONVERSION TO MEMORY (GB) RESOURCES ($f2(.)$)

VNF	Bandwidth requirement (Gbps)									
	1	2	3	4	5	6	7	8	9	10
NAT	1	1	1	1	1	1	1	1	2	2
FW	2	2	4	4	8	8	16	16	32	32
TS	1	2	4	4	8	8	8	8	16	16
WOC	1	1	1	1	2	2	2	2	2	4
IDPS	2	2	4	4	8	8	16	16	32	32
VOC	4	4	8	8	16	16	32	32	64	64

We consider 4 types of SCs with 6 types of NFs: Network Address Translator (NAT), Firewall (FW), Traffic Shaper (TS), WAN Optimization Controller (WOC), Intrusion Detection, and Prevention System (IDPS), Video Optimization Controller (VOC). Table VIII lists the SCs along with NF chaining requirements [20], [21]. For example, Web services require chaining of the following NFs, in this order: NAT, FW, TS, WOC, and IDPS. We also consider 4 difference types of services: Web services (search, etc.), Voice Over IP (VOIP) (audio), Video Streaming, and Online Gaming, with representative percentage in traffic.

Table IX shows conversion from network bandwidth requirement to compute (CPU) resource requirement, generated by extrapolating information from prior studies [20] and industry data [46].

Table X shows conversion from network bandwidth requirement to memory resource requirement inspired by prior studies [20] and industry data [46].

C. Impact of Prediction Methods on ORASC

For this part of the results (Figs. 7-9), we consider 100 SC request arrivals per hour, and the simulation runs for 24 hours. For this part of the study, we consider operator-driven auto-scaling (ORASC). To feed our ORASC algorithm, we use three prediction methods proposed in this study: i) DNN (with 95.1% accuracy), ii) LSTM: (with 95.7% accuracy), and iii) ML (Random Forest as the highest-performing ML algorithm with 96.5% accuracy).

We compare our proposed methods with three prior works: dynamic-threshold-based approach derived from cloud auto-scaling (DT) [10], a recent SC auto-scaling study (AR) [23] that uses auto-regression as prediction method, and an artificial-neural-network-based method (ANN) [35].

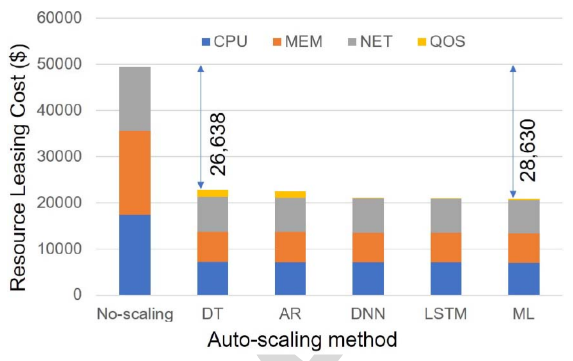


Fig. 7. Resource leasing cost vs. auto-scaling prediction methods.

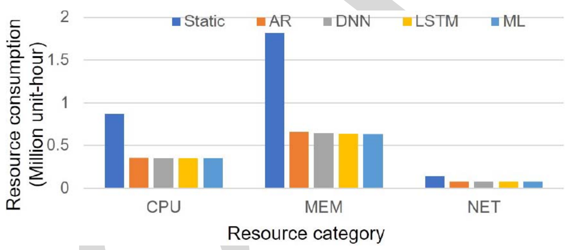


Fig. 8. Resource consumption compared in resource category and auto-scaling prediction methods.

Fig. 7 shows leasing cost saving for tenants using ORASC powered by different prediction methods. In addition to CPU, memory, and bandwidth leasing cost, we also consider the QoS penalty cost for tenants due to degraded service ($1 per hour). The left-most column shows total leasing cost for "No-scaling" method, where all leased resources are allocated all the time (hence no QoS cost for tenants). Then, auto-scaling using "Dynamic Threshold (DT)" based method saves $26,638 by auto-scaling the tenants' free resources. "Auto-Regression (AR)" based prediction method performs a little better than DT method, due to better prediction. Our proposed Random Forest algorithm (ML) yields lowest leasing cost (or savings of $28,630) for the tenants. The other methods (DNN and LSTM) outperforms the prior studies, and demonstrate cost savings very close to ML. Fig. 6 also reports the QoS degradation penalty incurred by under-provisioning of resources. DNN, LSTM, and ML methods show lower penalty compared to prior studies DT and AR.

Fig. 8 shows total resource consumption data breakdown for Fig. 7, in CPU, memory, and network units. As expected, DNN, LSTM, and ML perform better than prior study AR.

Fig. 9 compares impact of auto-scaling methods on different SC types. Here, we use different level of QoS cost for different SC types (e.g., Web = $3, VOIP = $2, Video = $1, and Gaming = $2.5). As the results show, impact of more accurate prediction (ML) is higher on the more QoS sensitive services.

D. Negotiation-Game-Based Auto-Scaling (NRASC)

For Figs. 10-12, we use the following set of values (uniformly) for tenants' 'Expected Refund' S_q^{α}: 0.2, 0.3, 0.4, 0.5,

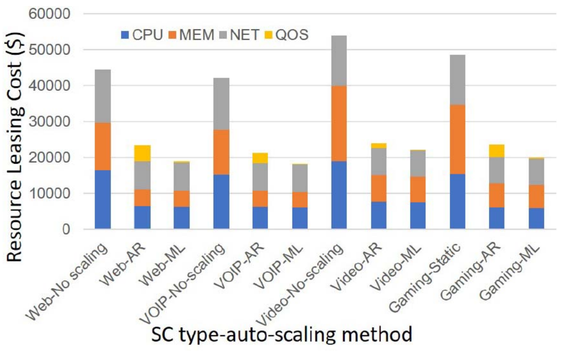


Fig. 9. Different SC types: resource leasing cost vs. auto-scaling prediction methods.

0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2. For maximum R, we use 1.0, indicating that, at maximum, the NO is willing to pay the original leasing cost as refund.

In this part of the study, we compare the impact of prediction methods on ORASC and NRASC, to demonstrate the benefits from negotiation. Our proposed highest-performing method 'ML' is compared with prior study 'AR' by using following three combinations: **i) O-AR:** ORASC algorithm powered by auto-regression-based prediction, **ii) O-ML:** ORASC powered by ML-based Random Forest algorithm. **iii) N-ML:** NRASC powered by ML-based Random Forest algorithm.

In Fig. 10, on the right two columns, 'Operator's cost savings' shows the win for the NO. Compared to operator-driven approach (O-ML), in negotiated approach (N-ML) NO pays much less ($14,617), as refund. In Operator-driven approach (O-ML), NO makes auto-scaling decision without considering different QoS levels of tenants, hence offering full refund to all the tenants. On the other hand, in negotiation-based approach (N-ML), the NRASC algorithm is aware of the QoS levels of the tenants. Hence, NO engages in a negotiation game with the tenants, so that NO can pay the least amount of refund (compared to full refund in O-ML). This results in lower cost for N-ML in Fig. 10.

On left, for negotiated approach (N-ML), tenant pays less leasing cost by gaining $23,641 from auto-scaling refund (compared to no scaling). Some service chains have higher 'Expected Refunds' (e.g., ER = 1.1, 1.2) than the maximum 'Offered Refund', as a result. Without considering QoS penalty, we observe that N-ML saves less money for tenants than the Operator-driven approach (O-ML). But, considering higher QoS penalty (e.g., 100 per hour for ER = 1.1 and 1.2), O-ML will fail to capture the importance of the sensitive service chains and auto-scale anyway, resulting in high QoS penalty. Hence, negotiated approach (N-ML) increases the tenants' happiness as well, by creating a win-win situation.

Figs. 11 and 12 show insights for Fig. 10. Fig. 11 compares released resources (unit-hour) by tenants (which leads to refund and reduced leasing cost). As discussed before, Operator-driven approach, powered by machine learning (O-ML) and powered by auto-regression (O-AR), releases more excess resources, compared to negotiation-based approach

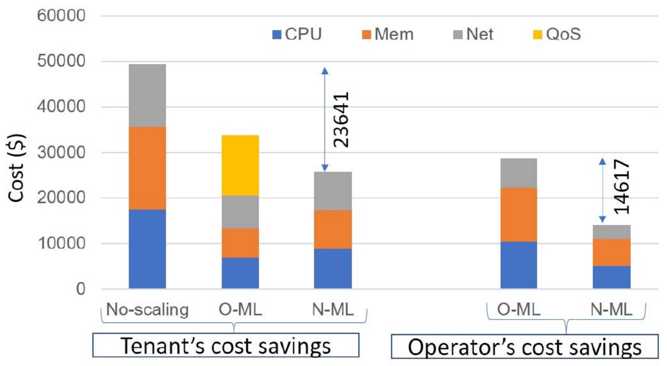


Fig. 10. Win-Win for Operator and tenants: cost vs. auto-scaling methods.

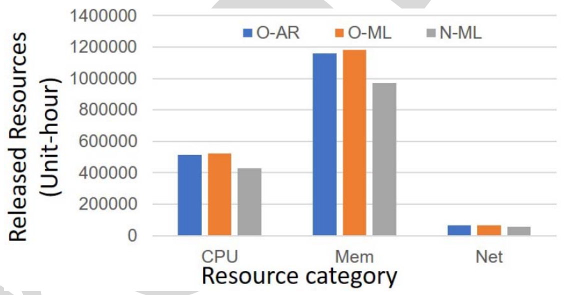


Fig. 11. Resource consumption compared in resource category and auto-scaling methods.

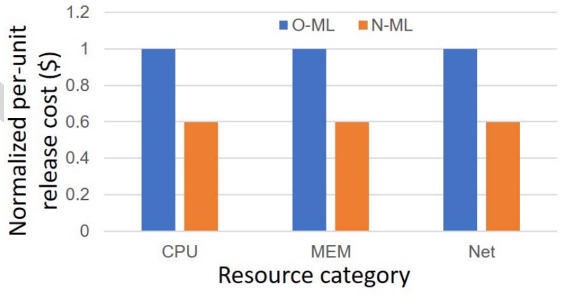


Fig. 12. Refund cost for operator vs. auto-scaling methods.

powered by ML (N-ML). But excess released resources from O-AR and O-ML come with excess QoS penalty, as shown in Fig. 10.

Fig. 12 shows normalized refund cost over different resource categories. We observe around 40% cost savings across all resource categories.

To further investigate the NO's cost benefit (associated to refund) due to negotiation (N-ML), Fig. 13 shows the impact of maximum offered refund in case of computing resources. At maximum offered refund = 0.5, to release 200,853 unit-hours of CPU resource, the NO pays for only 68,439 unit-hours equivalent refund, leading to payment of 34% of the original cost (hence, the NO can keep 66% of the leasing cost to itself). At higher value for maximum offered refund (e.g., 0.9), NO pays 54% of the original cost (hence, the NO can only keep 46% of the leasing cost to itself). We observe similar trends for memory and network resources as well.

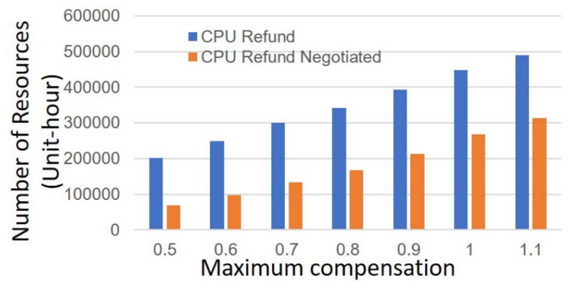


Fig. 13. Impact of maximum refund offered on NO's refund cost.

VII. Conclusion

Our study proposed two SC auto-scaling methods which benefit from ML prediction method. The ML classifier learns from historic data and shows promising accuracy (96.5%). Illustrative results explain different aspects of the proposed ML prediction model. Our proposed auto-scaling methods consider practical SC usecase scenarios with a backbone network and geo-distributed DCs/Edge-DCs. Numerical results show that our proposed prediction method yields lower leasing cost for network tenants compared to prior works. In addition, negotiation-game-based auto-scaling method reduces both tenant leasing cost and NO's refund cost, while respecting tenants' willingness to participate in the auto-scaling process. Future studies should explore more detailed operational and leasing costs.

References

[1] "Network functions virtualisation: Introductory white paper," ETSI, Sophia Antipolis, France, White Paper, 2012.

[2] L. Peterson *et al.*, "Central office re-architected as a data center," *IEEE Commun. Mag.*, vol. 54, no. 10, pp. 96–101, Oct. 2016.

[3] M. Mechtri, C. Ghribi, and D. Zeghlache, "A scalable algorithm for the placement of service function chains," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 533–546, Sep. 2016.

[4] P. Hong, K. Xue, and D. Li, "Resource aware routing for service function chains in SDN and NFV-enabled network," *IEEE Trans. Services Comput.*, early access, Jun. 22, 2018, doi: 10.1109/TSC.2018.2849712.

[5] J. Pei, P. Hong, K. Xue, and D. Li, "Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system," *IEEE Trans. Parallel Distrib. Syst.* vol. 30, no. 10, pp. 2179–2192, Oct. 2019.

[6] A. Gupta, M. F. Habib, U. Mandal, P. Chowdhury, M. Tornatore, and B. Mukherjee, "On service-chaining strategies using virtual network functions in operator networks," *Comput. Netw.*, vol. 133, pp. 1–16, Mar. 2018.

[7] *Amazon Web Sevices—Auto-Scaling*, Amazon, Seattle, WA, USA. Accessed: May 15, 2018. [Online]. Available: https://aws.amazon.com/autoscaling/

[8] S. Rahman, T. Ahmed, M. Huynh, M. Tornatore, and B. Mukherjee, "Auto-scaling VNFs using machine learning to improve QoS and reduce cost," in *Proc. IEEE Int. Conf. Commun.*, Kansas City, MO, USA, May 2018, pp. 1–6.

[9] K. Kanagala and K. C. Sekaran, "An approach for dynamic scaling of resources in enterprise cloud," in *Proc. IEEE 5th Int. Conf. Cloud Comput. Technol. Sci.*, Bristol, U.K., 2013, pp. 345–348.

[10] M. M. Murthy, H. A. Sanjay, and J. Anand, "Threshold based auto scaling of virtual machines in cloud environment," in *Proc. Int. Conf. Netw. Parallel Comput.*, 2014, pp. 247–256.

[11] C. Hung, Y. Hu, and K. Li, "Auto-scaling model for computing system," *Int. J. Hybrid Inf. Technol.*, vol. 5, no. 2, pp. 181–186, Apr. 2012.

[12] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *J. Grid Comput.*, vol. 12, no. 4, pp. 559–592, 2014.

[13] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proc. 8th Int. Workshop Middleware Grids Clouds e-Sci.*, 2010, pp. 1–6.

[14] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: Challenges and opportunities," in *Proc. 1st Workshop Autom. Control Datacenters Clouds*, 2009, pp. 13–18.

[15] A. Chandra, W. Gong, and P. Shenoy, "Dynamic resource allocation for shared data centers using online measurements," in *Proc. 11th Int. Conf. Qual. Service*, 2003, pp. 381–398.

[16] H. Mi, H. Wang, G. Yin, Y. Zhou, D. Shi, and L. Yuan, "Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers," in *Proc. IEEE Int. Conf. Service Comput.* Miami, FL, USA, 2010, pp. 514–521.

[17] W. Fang, Z. Lu, J. Wu, and Z. Cao, "RPPS: A novel resource prediction and provisioning scheme in cloud data center," in *Proc. IEEE 9th Int. Conf. Service Comput.*, Honolulu, HI, USA, 2012, pp. 609–616.

[18] H. Tang, D. Zhou, and D. Chen, "Dynamic network function instance scaling based on traffic forecasting and VNF placement in operator data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 3, pp. 530–543, Mar. 2019.

[19] T. Phung-Duc, Y. Ren, J. C. Chen, and Z. W. Yu, "Design and analysis of deadline and budget constrained autoscaling (DBCA) algorithm for 5G mobile networks," Sep. 2016. [Online]. Available: arXiv:1609.09368.

[20] A. Gupta, B. Jaumard, M. Tornatore, and B. Mukherjee, "A scalable approach for service chain mapping with multiple SC instances in a wide-area network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 529–541, Mar. 2018.

[21] M. Savi, M. Tornatore, and G. Verticale, "Impact of processing costs on service chain placement in network functions virtualization," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, San Francisco, CA, USA, 2015, pp. 191–197.

[22] H. Moens and F. D. Turck, "Customizable function chains: Managing service chain variability in hybrid NFV networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 13, no. 4, pp. 711–724, Dec. 2016.

[23] J. Duan, C. Wu, F. Le, A. X. Liu, and Y. Peng, "Dynamic scaling of virtualized, distributed service chains: A case study of IMS," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2501–2511, Nov. 2017.

[24] Y. Jia, C. Wu, Z. Li, F. Le, A. Liu, and Z. Li, "Online scaling of NFV service chains across geo-distributed datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 699–710, Apr. 2018.

[25] H. Yu, J. Yang, and C. Fung, "Elastic network service chain with fine-grained vertical scaling," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, UAE, Dec. 2018, pp. 1–7.

[26] A. N. Toosi, J. Son, Q. Chi, and R. Buyya, "ElasticSFC: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds," *J. Syst. Softw.* vol. 152, no. 2, pp. 108–119, 2019.

[27] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network support for resource disaggregation in next-generation datacenters," in *Proc. 12th ACM Workshop Hot Topics Netw. (Hotnets)*, Nov. 2013, pp. 1–7.

[28] *ECOMP (Enhanced Control, Orchestration, Management & Policy) Architecture White Paper*, AT&T Inc., Dallas, TX, USA. Accessed: May 10, 2018. [Online]. Available: http://about.att.com/content/dam/snrdocs/ecomp.pdf

[29] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," in *Proc. Conf. Emerg. Artif. Intell. Appl. Comput. Eng.*, 2007, pp. 3–24.

[30] E. Frank, M. A. Hall, and I. H. Witten, *The WEKA Workbench. Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. Burlington, MA, USA: Morgan Kaufmann, 2016.

[31] S. Rahman, A. Gupta, M. Tornatore, and B. Mukherjee, "Dynamic workload migration over backbone network to minimize data center electricity cost," *IEEE Trans. Green Commun. Netw.*, vol. 2, no. 2, pp. 570–597, Jun. 2018.

[32] *Energy-Efficient Ethernet Architecture Task Force*, IEEE Standard P802.3az. Accessed: May 15, 2018. [Online]. Available: ieee802.org/3/az/public/index.html

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *J. Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[34] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017.

[35] T. P. Oliveira, J. S. Barbar, and A. S. Soares, "Computer network traffic prediction: A comparison between traditional and deep learning neural networks," *Int. J. Big Data Intell.*, vol. 3, no. 1, pp. 28–37, 2016.

[36] S. Kraus, "Automated negotiation and decision making in multiagent environments," in *ECCAI Advanced Course on Artificial Intelligence*. Heidelberg, Germany: Springer, Jul. 2001.

[37] J. Alwen, A. Shelat, and I. Visconti, "Collusion-free protocols in the mediated model," in *Proc. Annu. Int. Cryptol. Conf.*, Heidelberg, Germany, Aug. 2008, pp. 497–514.

[38] J. Kattan and W. R. Vigdor, "Game theory and the analysis of collusion in conspiracy and merger cases," *George Mason Law Rev.*, vol. 5, no. 3, pp. 441–456, 1997.

[39] *Internet Traffic Data*, DataMarket, Reykjavík, Iceland. Accessed: May 15, 2018. [Online]. Available: https://datamarket.com

[40] *InfoGainAttributeEval*, WEKA, Hamilton, New Zealand. Accessed: May 15, 2018. [Online]. Available: http://weka.sourceforge.net/doc.stable-3-8/weka/attributeSelection/InfoGainAttributeEval.html

[41] *The Problem With Backpropagation*, Towards Data Science, San Francisco, CA, USA. Accessed: Nov. 15, 2019. [Online]. Available: https://towardsdatascience.com/the-problem-with-back-propagation-13aa84aabd71

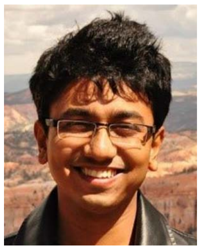
[42] S. Lang, F. Bravo-Marquez, C. Beckham, M. Hall, and E. Frank, "WekaDeeplearning4j: A deep learning package for Weka based on deeplearning4j," *Knowl. Based Syst.*, vol. 178, pp. 48–50, Aug. 2019.

[43] *The Fall of RNN/LSTM*, Towards Data Science, San Francisco, CA, USA. Accessed: May 30, 2020. [Online]. Available: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0

[44] *Google Cloud Pricing*, Google, Mountain View, CA, USA. Accessed: Nov. 1, 2019. [Online]. Available: https://cloud.google.com/compute/pricing

[45] *Google Fiber Pricing*, Google, Mountain View, CA, USA. Accessed: Nov. 1, 2019. [Online]. Available: https://fiber.google.com/cities/kansascity/plans/

[46] *Cisco Integrated Services Virtual Router Data Sheet*, Cisco, San Jose, CA, USA. Accessed: Nov. 5, 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/enterprise-network-functions-virtualization-nfv/datasheet-c78-736768.html

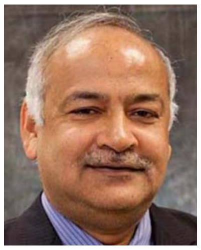


Sabidur Rahman (Graduate Student Member, IEEE) received the B.S. degree from the Bangladesh University of Engineering and Technology in 2011, and the M.S. degree in computer science from the University of Texas at San Antonio in 2014. He is currently pursuing the Ph.D. degree in computer science with the University of California at Davis. He also has research and development experience with AT&T Labs and Samsung Research and Development. His research interests include computer network virtualization, use of machine learning and data analytics to solve network research problems, and cost-efficient networking.

Tanjila Ahmed (Graduate Student Member, IEEE) received the B.S. degree in electrical electronics and communication engineering from the Military Institute of Science and Technology, Dhaka, Bangladesh, in 2010, and the M.S. degree in electrical engineering from the University of Texas at San Antonio in 2015. She is currently pursuing the Ph.D. degree in electrical and computer engineering with the University of California at Davis. She worked as a research intern with AT&T labs and a software engineer intern with Honeywell Inc., in their respective Machine Learning Development Team. Her research interests include datacenter networks, optical backbone network, and Internet-of-Things networks.

Minh Huynh (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from the University of California at Davis. He joined AT&T Labs working on network performance analysis and capacity planning in mobility networks and software-defined networks. His current role with Google, he is responsible for demand forecast modeling for the datacenter networks.

Massimo Tornatore (Senior Member, IEEE) is currently an Associate Professor with the Department of Electronics, Information, and Bioengineering, Politecnico di Milano, Italy. He also holds an appointment as an Adjunct Professor with the University of California at Davis, Davis, CA, USA, and as a Visiting Professor with the University of Waterloo, Waterloo, ON, Canada. His research interests include performance evaluation, optimization and design of communication networks (with an emphasis on the application of optical networking technologies), cloud computing, and machine learning application for network management. He has coauthored more than 300 peer-reviewed conference and journal papers (with 17 Best Paper Awards), two books and one patent, in the above areas. He is an Active Member of the Technical Program Committee of various networking conferences, such as INFOCOM, OFC, ICC, and GLOBECOM. He is a member of the editorial board of IEEE COMMUNICATION SURVEYS & TUTORIALS, IEEE COMMUNICATION LETTERS, *Photonic Network Communications* (Springer), and *Optical Switching and Networking* (Elsevier).



Biswanath Mukherjee (Fellow, IEEE) received the B.Tech. degree from the Indian Institute of Technology Kharagpur in 1980, and the Ph.D. degree from the University of Washington, Seattle, in 1987. He is a Distinguished Professor with the University of California at Davis, where he was Chairman of computer science from 1997 to 2000. He served a 5-year term on Board of Directors of IPLocks, a Silicon Valley startup company (acquired by Fortinet). He has supervised 77 Ph.D.'s to completion and currently mentors six advisees, mainly Ph.D. students. He has authored the graduate-level textbook *Optical WDM Networks* (Springer, January 2006). He is a winner of the 2004 Distinguished Graduate Mentoring Award and the 2009 College of Engineering Outstanding Senior Faculty Award at UC Davis. He is a co-winner of the Optical Networking Symposium Best Paper Awards at IEEE Globecom 2007, 2008, and 2019. He has served on the Technical Advisory Board of several startup companies, including Teknovus (acquired by Broadcom). He was General Co-Chair of the IEEE/OSA Optical Fiber Communications (OFC) Conference in 2011, the Technical Program Co-Chair of OFC'2009, and the Technical Program Chair of the IEEE INFOCOM'96 conference. He is an Editor of *Optical Networks* Book Series (Springer). He has served on eight journal editorial boards, most notably IEEE/ACM TRANSACTIONS ON NETWORKING and IEEE NETWORK. He has Guest Edited Special Issues of the PROCEEDINGS OF THE IEEE, the IEEE/OSA JOURNAL OF LIGHTWAVE TECHNOLOGY, the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, and IEEE COMMUNICATIONS.