Received: 15 December 2022 | Revised: 31 May 2023 | Accepted: 7 June 2023

IET Networks

DOI: 10.1049/ntw2.12090

ORIGINAL RESEARCH

Optimal resource optimisation based on multi-layer monitoring

Dimitrios Uzunidis[1] | **Panagiotis Karkazis**[2] | **Helen C. Leligou**[3]

[1]Department of Electrical and Electronics Engineering, University of West Attica, Egaleo, Greece

[2]Department of Information and Computer Engineering, University of West Attica, Egaleo, Greece

[3]Department of Industrial Design and Production Engineering, University of West Attica, Egaleo, Greece

Correspondence

Panagiotis Karkazis.
Email: p.karkazis@uniwa.gr

Funding information

Special Account for Research Grants (ELKE)

Abstract

The satisfaction of the Quality of Service (QoS) levels during an entire service life-cycle is one of the key targets for Service Providers (SP). To achieve this in an optimal way, it is required to predict the exact amount of the needed physical and virtual resources, for example, CPU and memory usage, for any possible combination of parameters that affect the system workload, such as number of users, duration of each request, etc. To solve this problem, the authors introduce a novel architecture and its open-source implementation that a) monitors and collects data from heterogeneous resources, b) uses them to train machine learning models and c) tailors them to each particular service type. The candidate solution is validated in two real-life services showing very good accuracy in predicting the required resources for a large number of operational configurations where a data augmentation method is also applied to further decrease the estimation error up to 32%.

KEYWORDS

machine learning, next generation networking, performance estimation, software defined networking

1 | INTRODUCTION

Engineers have tried to find the optimal balance between the devoted computational and networking resources per service and the desired QoS since the beginning of telecommunication systems. Although it is obvious that overallocation of resources ensures higher QoS, this approach has a negative impact on the economics of infrastructure operators. In contrast, resource under-allocation can result in a violation of the Service Level Agreement (SLA), which is considered unacceptable. The definition of the "critical point," where the allocated resources ensure the agreed SLA with zero underutilisation, has proven difficult. So, in most cases, the engineers have resorted to resource over-allocation whilst trying to minimise the distance between this over-allocation from the "critical point".

At the same time, continuous efforts towards more dynamic, intelligent, and sophisticated network management solutions have resulted in new concepts, like the Software Defined Networking (SDN) and Network Function Virtualisation (NFV) technologies, which are currently at their hype [1]. In the NFV context, any Network Service (NS) (ex. Firewall, CDN, router etc.) can be broken down into a chain of Virtual Network Functions (VNF), which can be deployed on the same or different virtual infrastructures (NFVI) offering high configuration flexibility to network operators. Meantime, network operators struggle to maximise the benefits from their infrastructures [2, 3]. To do so, they need mechanisms that can predict in a reasonable time frame the necessary resources for each running service per level of QoS. This requires knowledge of the profile of each deployed service, which is on its own another challenge as new types of services emerge every day. This approach creates a dynamic and extremely complex environment in which the resource allocation problem becomes more complex than ever, since decisions must be made faster and at a finer level.

Another critical issue that must be considered is the heterogeneity of the modern datacenters infrastructure, in combination with the virtualisation technologies that are used. The different types of equipment (i.e. physical servers, switches, routers, etc.) are managed by hypervisors that provide virtual network and computational resources (i.e., virtual machines, virtual networks, containers, etc.). This mandates the monitoring and management of the allocated resources at all three layers: the physical, the virtual, and the service/application layer. The monitoring of the first two layers is critical, as it can provide knowledge about the impact of service workload on the system performance metrics, for example, CPU, disk, memory usage, and can aid to avoid operating beyond the system's "critical point". Next, the monitoring of service/application layer performance metrics, for example, service time, is also critical as the performance metrics can be directly related to QoS.

In this complex scene, Machine Learning (ML) based tools can lead to optimal management of the available resources [4–8]. Assuming that any type of service has specific characteristics (i.e., different injection patterns), the service profile can be automatically derived via ML techniques and thus, the system can be equipped with the ability of "learning" new service profiles as they emerge. These profiles are mandatory for the definition of the resources that should be allocated to satisfy the agreed QoS level. So, an automated process of service profiling and network performance prediction can assist the network operators to take a step forward in improving the utilisation of their infrastructures without sacrificing QoS and/or service availability.

In prior art, ML and rule-based methods have been used to predict the necessary resources based on data primarily collected from virtual infrastructures [4, 6]. In particular, in [4], four cases namely, a virtual router, a switch, a firewall and a cache server were deployed and various algorithms spanning from linear regression to artificial neural networks were exploited to estimate CPU usage, packet loss and cache response time under different operational conditions. Further, in [6], ML techniques were trained to predict the workload impact on CPU, total delay and number of VNF instances for different services attaining very good accuracy along with low estimation time. In the aforementioned works, only the data of the virtual layer were exploited, while approaches that monitor, collect, and exploit data from all implementation layers, including physical, virtual, and service, are still in their infancy [9–13] and under consideration. In particular, [9, 10] adopt multi-layer monitoring and data collection mechanisms, however, without considering the use of ML, while in [11–13], ML is adopted and the algorithms were optimised in a relatively small number of data. In general, we could say that the validity of the approaches of the literature is confined to only relatively small number of services and infrastructures with small capacities, for example, with two real servers and VMs and only one virtualisation technology. Also, fewer performance indicators are predicted and in a much smaller set of service configurations, which limits the adoption of more advanced ML techniques to improve the achievable accuracy.

The motivation of our work is to address the open challenge which is elaborated in the next section and is the minimisation of assigned resources for each service instance while ensuring the QoS. To address this challenge, we introduce a novel mechanism that exploits the most recent open-source network monitoring technologies, and we combine them with ML techniques to automate the process and bring the deployment parameters (resource allocation) closer to the "critical point". To examine the feasibility of the proposed approach, we perform service profiling and performance predictions in two well-known services, which are Hadoop and a backend service, by collecting and analysing an extensive list of monitoring metrics from physical and virtual infrastructure namely CPU/memory usage, network throughput, etc., and performance metrics from the running services. This realises the concept of multi-layer monitoring and holistic optimisation, as we monitor, collect, and exploit data from all three layers, namely physical, virtual, and service. The modelling methodology in these two selected services is of relatively low complexity, allowing to obtain explainable and meaningful results that validate the efficacy of the candidate approach.

In the rest of the paper, in section II, we elaborate the challenge we try to address, in section III, we introduce the candidate solution, in section IV, we describe the sandbox environment that we deployed to validate the proposed mechanism and in section V, we discuss the obtained results. Finally, section VI concludes the paper.

2 | PROBLEM DEFINITION

The most important challenge for service providers is to minimise the number of allocated resources while ensuring the agreed QoS. In the context of NFV, where various VNFs are chained together to form an NS, the prediction of the exact amount of the required resources, for example, CPU, memory, service time etc., per service demand, at any time of its lifecycle is not an easy process mainly a) due to the dynamic nature of user's demands which require the infrastructure that hosts the NS to be adapted in very short time intervals and b) due to the vast number of performance metrics that need to be monitored, from various layers of the infrastructure over the entire service lifecycle.

To address this challenge, two main procedures are required. The first incorporates monitoring probes to monitor and collect data from three layers of the tested system. The second is based on the collected data to train algorithms that can predict the required system/network resources for a broad gamut of operational scenarios and workloads, such as number of requests, size of each request etc. An example of the first procedure is illustrated in Figure 1. This figure illustrates the impact of (a) small input workload and (b) large input workload on various performance metrics of the tested environment that hosts a NS. In the case (b), we can observe that two metrics, one in virtual and one in physical layer, exceed the usage threshold of 100% designating a clear "breaking point" of the tested environment that can potentially lead to QoS violation.

3 | PROPOSED SOLUTION

The proposed architecture can be deployed in any datacentre, which provides virtual computational and network resources and in its typical form includes a) multiple hardware devices at the physical (equipment) layer, b) virtual resources (VMs, containers, SDN controllers) at the virtualisation layer and c) application layer entities, which consist of the diverse services that run on this node. In this work, the specified layers are not related to the OSI-compliant layers. The proposed architecture

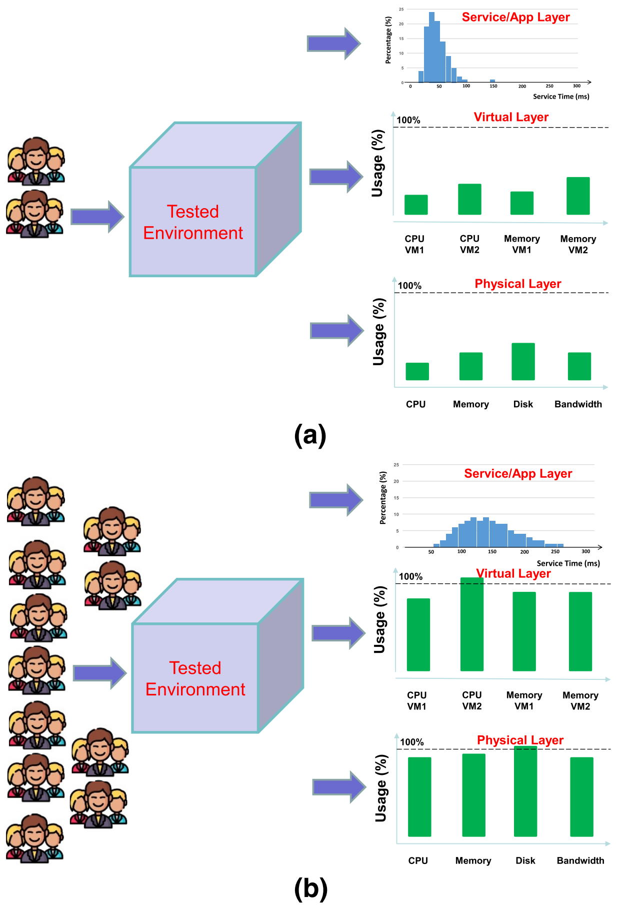

FIGURE 1 Monitoring of system metrics in all three layers for (a) small workload, for example, small number of requests and (b) large workload, for example, large number of requests.

consists of two main components: (i) a cross-layer monitoring service and (ii) a data analysis service.

The cross-layer data collection is realised by collecting data from the "monitoring probes" at each of the three layers and delivering them to the monitoring server (Figure 2). In more detail, regarding the application layer, the performance of each service can be easily measured by legacy monitoring solutions as many application servers, databases, etc. already provide monitoring tools to expose performance metrics. The information coming from the application layer is useful to define the break points of a service, but in most cases, the correlation of a specific failure with the actual cause is not straight forward.

The analysis service exploits the collected data to train various ML regression algorithms, spanning from linear regression to deep neural networks, which can accurately predict the relation between specific workload (affected by the number of concurrent requests, packet size, etc.) and the required resources (CPU, memory, network bandwidth, etc.). The overall accuracy depends on the adopted ML algorithm as well as on the dataset because, in principle, the accuracy improves with time, as the number of collected (and used for training) data is increasing while the service runs. The continuous training of the algorithms (apart from improved accuracy) enables the system to profile any new service in the

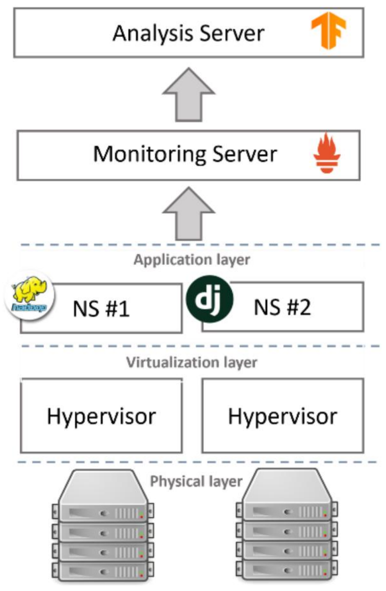

FIGURE 2 Data collection from the application, virtualisation and physical layer.

future. As a result, the impact of the workload parameters can be quantified for any possible resource configuration scenario.

4 | EVALUATION IN A REAL LIFE TESTBED

4.1 | The sandbox testbed

In order to validate the proposed architecture, we used a sandbox environment consisting of two different physical servers connected through a local ethernet network. The technical specifications for the physical servers are as follows: a) CPU: 4 CPUs x Intel(R) Xeon(R) (CPU E3-1220 v6 @ 3.00 GHz), b) RAM: 16 GB DDR4, c) HDD 8 TB d) NETWORK: 2 x 1GbE LOM, and the networking device that was used is the Cisco SG250 18 port Gigabit switch.

In the first server, we deployed all the components of the proposed architecture using docker-composed scripts, and on the second, following the same approach, we deployed the under-test services and the monitoring agents (cAdvisor and Netdata. io). All the software tools were based on the latest versions of container images from their open-source projects (i.e., Prometheus. io, Netdata. io, cAdvisor, Apache Spark, etc.) that make the proposed architecture reliable and re-deployable. Furthermore, the adoption of the Prometheus monitoring server prepares the proposed solution to integrate and collect monitoring data from Kubernetes clusters.

After the setup of the proposed architecture, two separate group of tests were performed using a) a Hadoop cluster and b) a production set up of the backend service consisting of a set of three micro-services (application server, database, and web server). The selection of the specific service types was based on the fact that both Hadoop and backend implementations are widely used in the industry. The components that were used in the proposed architecture are opensource Cloud Native implementations. In more detail:

1) Monitoring Server:

The core component of the monitoring framework is Prometheus. io server which stands as the central point of event monitoring, storage and alerting. All metrics from the three layers are collected, using a HTTP pull model, and stored in a timeseries database. Some of the key features that make this server suitable for the proposed architecture are (a) use of a flexible query language (PromQL), which eases the interconnection with external systems (ex. Analytics module); (b) existence of many opensource implementation (exporters) for exposing monitoring metrics from various applications, and it is also quite easy to create new ones; (c) autonomy as there is no reliance on complex distributed storage mechanisms; and (d) new monitoring targets can be easily added via reconfiguration or using the file-based service discovery mechanisms.

Let us now turn our attention to the monitoring/probing mechanisms of the three layers. For the *physical layer monitoring*, or the monitoring of the physical servers, we chose Netdata. io, which is a powerful tool designed to collect a huge list of metrics from every system and application in real-time. For the v*irtual machines monitoring*, the micro-services under test are hosted in different VMs and to monitor their performance we used a Netdata plugin specially designed to collect metrics from vSphere ESXi. So, Prometheus collects the target metrics from the Netdata for both physical and virtual resources. Regarding the containers, we can easily integrate the appropriate tools (ex. cAdvisor etc.) or we can directly monitor Kubernetes cluster using Prometheus. For the application layer, to monitor the performance of the service under-test, we used two additional tools compatible with the monitoring server (Prometheus). The first one is an additional Django application that collects performance metrics from the server and exposes them to the Prometheus server. The second one is the Postgres exporter which also collects performance metrics from the DB and exposes them to the Prometheus.

2) Analysis Server:

We employed 4 ML algorithms to find the relationship between the workload parameters and the target system metrics. These ML algorithms are: Deep Neural Network (DNN), Decision Tree (DT), Random Forest (RF), and k-Nearest Neighbours (k-NN). They were selected from a long list of ML methods as the most suitable for our problem, as they provided high modelling accuracy along with very low computational time. Below, we briefly present the operation of the 4 ML methods.

a) k-Nearest Neighbours

K-NN algorithm is trained as follows. First, it calculates the Euclidean distances between a new data point and each of the data points of the training set as follows:

$$d(y_i, \hat{y}) = \sqrt{(x_{i,1} - x_1)^2 + (x_{i,2} - x_2)^2 + \ldots + (x_{i,n} - x_n)^2} \quad (1)$$

where the dependent variable y_i is the target metric and the independent variables x_1, x_2 …, x_n are the input features, for example, number of files and file size. Then, it selects the $\boldsymbol{k}$ y points with the smaller Euclidean distance to calculate the predicted $\hat{y}_i$ value by taking their average.

b) Decision Tree

DT leads to a data organisation in a tree structure which begins from the root node towards the leaf nodes, by asking each time a question which has a binary answer, for example, *yes/no*. Every time a question is answered, new branches are created and the feature space is segregated into disjoint regions. The selection of the optimal feature in each node and the proper question are usually based on the feature with the highest information gain. As a consequence, by selecting the proper number of tree depth, a decision tree with a large number of nodes that can provide estimations for the value y can be created.

c) Random Forest

RF in essence comprises an ensemble of decision trees. This forest is usually trained using the bagging method. The main advantage of RF over DT is that multiple trees are employed to provide a result instead of a single tree, as a large number of uncorrelated trees operating as an ensemble can provide a greater accuracy compared with an individual tree.

d) Deep Neural Network

DNN is a technique based on layers of inter-connected neurons—in our case we exploit fully connected layers—that are employed to solve regression problems accurately. One key advantage of DNN is its ability to learn complex nonlinear relationships between the input data (independent variables $x_1, x_2 \ldots, x_n$) and the output data (dependent variable y). DNN uses an activation function in each neuron in order to ignite it, which in our case is the Rectified Linear Unit (ReLU). One main disadvantage of DNN is that it usually requires a large number of training data in order to provide accurate results at its output.

For our ML predictions, we used Python (version 3.5) as programming language, and specifically the Numpy library for matrix multiplications, data preprocessing and segmentation, the scikit-learn library for implementing the ML algorithms, and the Keras high-level neural networks library using as backend the Tensorflow library (version 2.0.0).

4.2 | Evaluation process

As an evaluation test for Hadoop, we exploited "testDFSIO" in order to stress the deployed implementation while monitoring data from all three layers, particularly from the "testDFSIO" (service layer), hardware and virtual layers. The "testDFSIO" is used to test the performance of NameNode and network components in Hadoop Distributed File System (HDFS) and was selected as it is an important test for the Hadoop cluster providing an overall performance evaluation of the examined service and a fast impression of how efficient the cluster is in terms of IO. More details about "testDFSIO" can be found in [11].

To stress the backend service, we employed Apache Bench, which fits better to the under-test service and we used it to transmit concurrent requests towards the examined service. After the stressing testing procedure, a large number of metrics are collected from all three layers of the tested environment. At the end of the test, the monitoring server collects information from the test bench tool, which are related with the performance of the under-test service (ex. statistics regarding the execution time of the requests, number of requests per sec etc.).

After data collection in both sandbox implementations, the number of recorded data are used to feed the ML algorithms which, after the proper training, can provide predictions for each resource type. For modelling purposes, we employed the four aforementioned ML algorithms to mathematically relate the input parameters, for example, the number of files and file size, to the monitored metrics, for example, CPU and disk usage.

5 | ML-AIDED PERFORMANCE PREDICTIONS

5.1 | The backend service

5.1.1 | Profiling of critical system metrics

In order to highlight the need for data analysis coming from multiple layers, we present the impact of the service stress test to physical, virtual and service layer. In Figure 3, we present the evolution of three metrics for different number of concurent requests: the blue line represents the CPU usage of the VM#3 (Django/Nginx micro services), the green line denotes the CPU usage of VM#4 (Postgress) and the red one describes the CPU usage of the physical server. Next, in Figure 4, we illustrate the mean execution time of the requests, for 50%, 90% and 99% of the requests. These metrics were selected from a wide list offered by the tool as the most recognised/comprehensive for our audience. The stress test consisted of a total number of 10,000 HTTP POST requests with a JSON payload of 3 KB, containing 10,000 records. The same test was executed with incremental number of concurrent requests to push the service to the limit.

As it is evident from Figure 3, when the concurrency level is about 10, the CPU usage of the web server (hosted in VM#3) reaches its maximum, designating a system "critical point", whilst, at the same time, the CPU usage of the DB, which is hosted in VM#4, is well below 100%. Thus, a different CPU allocation scheme between the two VMs can extend the system "critical point" and allow servicing a larger number of concurrent users.

Turning our attention to the request execution time, shown in Figure 4, this can provide insights about the efficiency of the deployed scenario from a service perspective. The request execution time increases with the number of concurrent requests, as expected, as the requests experience longer waiting times to be executed. Moreover, 99% of the requests (triangles, purple) are served in less than a 100 ms time frame when 4 concurrent requests are considered, while half of the requests (rectangular, light blue) are served in less than 100 ms even when 10 concurrent requests request for service. The knowledge of this time is important for a service/network provider especially in the context of Guaranteed Reliable Experience category of services [14], as this time can be added to the end-to-end delay, and if it exceeds a pre-specified threshold can potentially lead to a QoS violation. As Figure 4 shows only a snapshot of these four metrics for different concurrency levels, the service provider must be able to dynamically predict the evolution of the most critical performance metrics under many different operating conditions and ML can significantly alleviate this task.

5.1.2 | Performance predictions using machine learning

In this section, we exploit the benefits of the ML analysis to predict the behaviour of a backend service, in order to act proactively to any potential change and ensure the requested QoS. In Table I, we present a quantitative comparison between the 4 ML algorithms that were employed. The number of

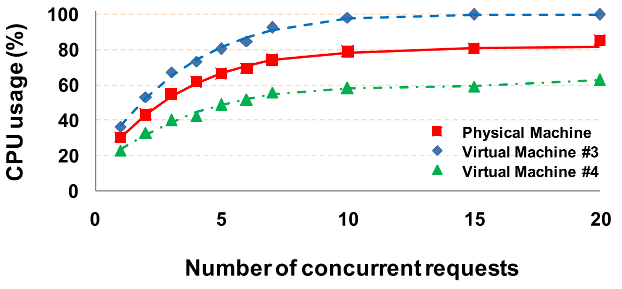

FIGURE 3 Impact of the number of concurrent requests on virtual and physical layer metrics.

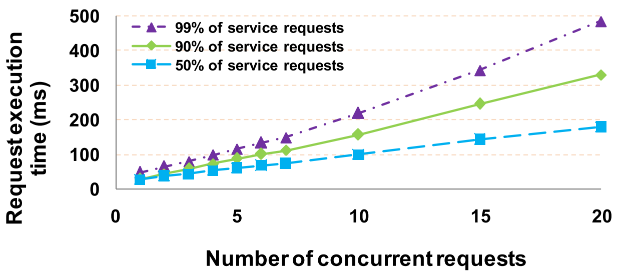

FIGURE 4 Impact of the number of concurrent requests on an app/service layer metric.

measured data points used to feed the ML algorithms were 500 for each metric. We consider this number of data points sufficient and as a consequence we omit data augmentation in this case. The data points were attained by iterating over the following parameters:

Number of concurrent requests:

$$[1, 2, 3, 4, 5, 6, 7, 10, 15, 20]$$

Number of records in the DB:

$$[1, 2, 3, 4, 5, 10, 20, 50, 100] \cdot 1000$$

Record size (kB):

$$[0.18, 1, 3, 5, 7]$$

The similar procedure with the previous case in [11] is followed here regarding the split of data in three sub-sets and the training of the ML methods. The optimal hyperparameters for each ML method are calculated using grid search in the validation set. Table I tabulates the percentage error of the four algorithms using the optimal hyperparameters as well as using another hyperparameter value for comparison purposes. As it is evident from Table I, the DT and RF outperform all other ML algorithms, due to their ability to approximate functions with floors with great efficiency (e.g. a floor in CPU usage is 100%). This is more evident in the most difficult-to-predict metric, which is the Network Bandwidth, as it reaches a floor when the CPU usage reaches 100% (i.e. after 10 concurrent requests, in our case).

RF algorithm attains a maximum error of 6.7% in all five metrics, outperforming DT due to the use of multiple trees. Further, DNN exceeds 10% which can be attributed to the relatively small number of training samples (300 in each case). Next, k-NN shows comparable performance when k is set to either 2 or 3 and the error is less than 15% in four out of five cases. Finally, the number of estimators has marginal effect on the predicted accuracy when RF is employed, as the accuracy improvement is no greater than 1%.

The aforementioned analysis reveals that ML algorithms can predict the performance of critical system metrics in the backend service with very good accuracy, less than 6.7% average error in all six metrics, for a large number of different operational conditions, using only a small fraction of these conditions.

5.2 | The hadoop case

In this section, we discuss the results of the ML analysis on the six metrics CPU usage, Disk Usage (physical layer), CPU usage, Memory Usage (virtual layer), Throughput, Average IO rate (service/app layer). The data analysis is highly advantageous for an SP as it can a) predict the behaviour of the examined system for any combination of critical system features, such as the number of files and file size not just those that have been previously tested, and b) designate additional system breaking points to those identified during the profiling phase, for example, the CPU usage on the node manager. This knowledge can be exploited by the SP in two ways. First, the SP can gain a clear knowledge about the breaking points of his deployed system architecture, and avoid operating his system at these points. Second, the prediction of the required resources under different workload, if it is joined with a resource allocation mechanism, can lead to an optimal use of the available resources, directly offering a cost and energy consumption reduction.

The initial dataset comprises 55 data points which were collected using the write test "testDFSIO" and are used to feed the ML algorithms. The results of 6 ML algorithms in the six metrics can be found in [11], confirming that ML algorithms can accurately predict the performance of critical system metrics, even when a minimal fraction of different system conditions is available.

In this work, we progress beyond our previous work and perform data augmentation to enrich the dataset in order to improve the prediction accuracy. In particular, we consider various curve fitting methods spanning from third order polynomials to logarithmic functions, tailored to each specific metric, for example, fitting the curves of Figure 3 of [11]. The augmented data points were attained considering combinations of the following parameters:

Number of files $\in$ [1,25].

File size (MB) $\in$ [100,1000].

Using this methodology, the dataset points are increased from 55 to 440 for all six metrics. As an ML algorithm and in order to examine the impact of data augmentation we employed k-NN algorithm as it attained the lowest error in the two service layer metrics, which was shown in [11] that they were the most difficult to predict. In particular, the main advantage of using k-NN in case of data augmentation is that the number of data points which have a smaller Euclidean distance are increased, compared with the initial dataset, leading to more accurate predictions.

It is worth mentioning that the augmented data have been exploited to improve the training and validation sub–sets only, while as testing cases, we considered only the initial 55 cases. During training and evaluation procedure, a circular rotation between the training, validation and testing cases was performed. The optimal value of k was found using the validation set and was equal to 2 for the physical and virtual layer metrics and equal to 1 for the service layer metrics. In the final step, the test sub-set was used to perform the ML predictions and then the predicted values were compared against the actual ones for each metric. The prediction error was calculated using the average absolute relative percentage error in order to attain comparable results across all the metrics and ML methods, as follows:

$$\frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i} \cdot 100\% \tag{2}$$

where y_i is the value of the ith point which was measured using our three-layer architecture, $\hat{y}_i$ is the predicted value using k-NN and n is the total number of points.

TABLE 1 Prediction error attained with 4 ML algorithms when they are applied to five system metrics.

	DT		RF		k-NN		DNN	
Hyper-parameters	**Tree depth**		**# of estimators**		**# of –k neighbors**		**# of neurons/layer, # of layers**	
Value	4	6	3	5	2	3	4, 2	8, 2
CPU usage (%) VM3	6.5%	4.6%	4.32%	3.9%	8.4%	8.6%	12.2	10.8%
CPU usage (%) VM4	6.2%	8.1%	7.3%	6.6%	11.8%	12.5%	18%	15.5%
CPU usage (%) PM	4.5%	3.1%	3.4%	2.7%	7.7%	8%	11%	9.9%
Value	6	8	10	20	2	3	8, 2	16, 2
Network bandwidth	10.9%	7.8%	7.2%	6.7%	28.7%	30%	30.4%	31.1%
Value	8	10	5	10	2	3	4, 2	8, 2
Execution time 90%	7.6%	6.4%	6.1%	5.6%	11.3%	11.9%	14.2%	12.2%

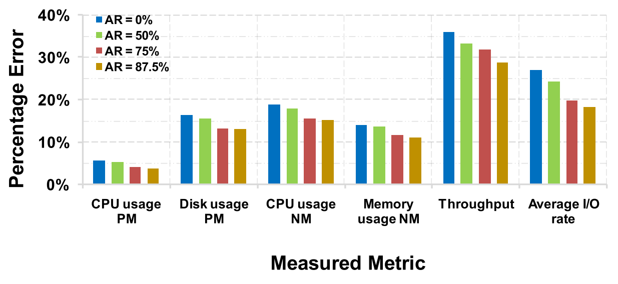

FIGURE 5 Impact of data augmentation on six studied metrics when k-NN algorithm is employed.

The impact of data augmentation is illustrated in Figure 5. As an Augmentation Ratio (AR) we introduce the following metric:

$$AR = \frac{Augmented\ Data}{Total\ Training\ Data} \cdot 100\% \qquad (3)$$

where the *augmented data* parameter includes the number of augmented data in the training set and the *total training data* parameter contains a) the number of augmented data and b) the “original” data points in the training set.

As can be seen in Figure 5, data augmentation reduces the prediction error in all six metrics. In particular, when an AR of 87.5% is considered, the error reduction is 1.80%, 3.32%, 3.71%, 2.95%, 7.28% and 8.77% in the CPU usage, Disk Usage, CPU usage, Memory Usage, Throughput and Average IO rate, respectively. This means that up to 32% lower error compared with the case of zero augmentation ratio can be attained, showing that data augmentation using curve fitting can be a capable solution, especially when the number of collected data is small. Finally, this dataset size increase can allow the use of more complex ML algorithms, such as deep neural networks, which require a greater number of input data and have the potential to provide an even higher prediction accuracy. Based on the encouraging results presented here, this approach could be adopted towards realising the Zero touch network & Service Management (ZSM) approach under standardisation in ETSI [15]. Automating the self-management of the network is at the core of this vision and the adoption of our proposed approach would definitely contribute to this direction. Namely, combining our approach with federated learning architectures like the one presented in [16], where security is also addressed, would contribute to the creation of automated self-managed network infrastructures.

6 | CONCLUSIONS AND FUTURE WORK

We have developed a solution that first performs multi-layer monitoring of an infrastructure that hosts a network service. Second, it exploits the collected data to feed various ML algorithms aiding to reveal potential “critical points” of the system, which may cause system failure, and also to identify the exact cause that is responsible for each failure. This is a very important knowledge for service providers as it can trigger them to act proactively to avoid the system's “critical points”, ensuring an optimal utilisation of their resources while guaranteeing a certain level of QoS. The candidate approach is based on open-source tools that enable automated service profiling and performance predictions using ML and has been validated in two well-known services, namely Hadoop and a backend service, showing very good accuracy in both cases. Finally, via using a data augmentation technique, the estimation error could be reduced by up to 32% compared with the case without data augmentation, showing that sufficient accuracy can be attained even in cases with a limited amount of available data.

In future, we plan to test our implementation with services in additional representative areas, such as media apps, security, etc. In addition, we want to train online ML methods for new NSs, as well as combine the proposed architecture with some of the open-source MANO frameworks to create a totally autonomous management system fit for deployment in the NFV environment. Another research avenue to pursue is to address open research challenges with respect to next generation computing proposed in [17], where some indicative are as follows: exploit ML approaches for security vulnerabilities forecasting in the fog layer, employ ML methods for enhancing the delay and reaction time of tasks in serverless computing for Internet of Things applications, train ML for the optimisation of edge systems leveraging vast number of data while maintaining the operational efficiency and estimation time, provide benefits using ML in Serverless systems from threat mitigation strategies, etc.

AUTHOR CONTRIBUTION

Dimitris Uzunidis: Writing – review & editing. **Panagiotis Karkazis**: Writing – review & editing. **Helen C. Leligoul**: Writing – review & editing.

ACKNOWLEDGEMENTS

This study has received financial support from the Special Account for Research Grants (ELKE) of University of West Attica.

CONFLICT OF INTEREST STATEMENT

None.

DATA AVAILABILITY STATEMENT

Data available on request from the authors.

REFERENCES

1. Peuster, M., et al.: Introducing automated verification and validation for virtualized network functions and services. IEEE Commun. Mag. 57(5), 96–102 (2019). https://doi.org/10.1109/mcom.2019.1800873
2. Alvarez, F., et al.: An edge-to-cloud virtualized multimedia service platform for 5G networks. IEEE Trans. Broadcast. 65(2), 369–380 (2019). https://doi.org/10.1109/tbc.2019.2901400
3. Giannakopoulos, I., et al.: PANIC: modeling application performance over virtualized resources. In: IEEE International Conference on Cloud Engineering, pp. 213–218. Tempe (2015)
4. Van Rossem, S., et al.: Profile-based resource allocation for virtualized network functions. IEEE Trans. Netw. Serv. Manag. 16(4), 1374–1388 (2019). https://doi.org/10.1109/tnsm.2019.2943779
5. Van Rossem, S., et al.: Optimized sampling strategies to model the performance of virtualized network functions. J. Netw. Syst. Manag. 28(4), 1482–1521 (2020). https://doi.org/10.1007/s10922-020-09547-8
6. Schneider, S., et al.: Machine learning for dynamic resource allocation in network function virtualization. In: Proceedings of 2020 6th IEEE Conference on Network Softwarization (NetSoft), pp. 122–130. Ghent, Belgium (2020)
7. Zafeiropoulos, A., et al.: Karl H "benchmarking and profiling 5G verticals' applications: an industrial IoT use case". In: 2020 6th IEEE Conference on Network Softwarization (NetSoft), pp. 310–318. IEEE (2020)
8. Mahsa, M., Ahmadi, M., Nikbazm, R.: Comparison of machine learning techniques for VNF resource requirements prediction in NFV. J. Netw. Syst. Manag. 30(2022), 1–29
9. Nam, J., Seo, J., Shin, S.: Probius: automated approach for VNF and service chain analysis in software-defined NFV. In: Proceedings of the Symposium on SDN Research (SOSR '18). Association for Computing Machinery, New York (2018). Article 14, 1–13
10. Trakadas, P., et al.: Comparison of management and orchestration solutions for the 5G era. J. Sens. Actuator Netw. 9(1), 4 (2020). No 4. https://doi.org/10.3390/jsan9010004
11. Uzunidis, D., et al.: Intelligent performance prediction: the use case of a Hadoop cluster. Electronics 10(21), 2021 (2690). https://doi.org/10.3390/electronics10212690
12. Karkazis, P.A., et al.: Intelligent network service optimization in the context of 5G/NFV. Signals 3(3), 587–610 (2022). https://doi.org/10.3390/signals3030036
13. Karkazis, P., et al.: Design challenges on machine-learning enabled resource optimization. IT Professional 24(5), 69–74 (2022). 1 Sept.-Oct. 2022. https://doi.org/10.1109/mitp.2022.3194129
14. Uzunidis, D., et al.: Fifty years of fixed optical networks evolution: a survey of architectural and technological developments in a layered approach. Tele.com (NY) 3(4), 619–674 (2022). https://doi.org/10.3390/telecom3040035
15. Zero-touch network and Service Management: Etsi - ZSM - zero touch network & service management. https://www.etsi.org/technologies/zero-touch-network-service-management (2022)
16. Short, A.R., Orfanoudakis, T.G., Leligou, H.C.: Improving security and fairness in federated learning systems. Int. J. Netw. Secur. Appl. 13(No. 6), 37–53 (2021). https://doi.org/10.5121/ijnsa.2021.13604
17. Singh, G., et al.: AI for next generation computing: emerging trends and future directions. Internet of Things 19, 100514 (2022). https://doi.org/10.1016/j.iot.2022.100514

How to cite this article: Uzunidis, D., Karkazis, P., Leligou, H.C.: Optimal resource optimisation based on multi-layer monitoring. IET Netw. 1–8 (2023). https://doi.org/10.1049/ntw2.12090