*electronics*

MDPI

*Article*

# Intelligent Performance Prediction: The Use Case of a Hadoop Cluster

**Dimitris Uzunidis [1,*], Panagiotis Karkazis [2], Chara Roussou [3], Charalampos Patrikakis [1] and Helen C. Leligou [3]**

[1] Department of Industrial Design and Production Engineering, University of West Attica, 12241 Athens, Greece; bpatr@uniwa.gr
[2] Department of Information and Computer Engineering, University of West Attica, 12243 Athens, Greece; p.karkazis@uniwa.gr
[3] Department of Electrical and Electronics Engineering, University of West Attica, 12241 Athens, Greece; auto45800@uniwa.gr (C.R.); e.leligkou@uniwa.gr (H.C.L.)
* Correspondence: duzunidis@uniwa.gr

**Abstract:** The optimum utilization of infrastructural resources is a highly desired yet cumbersome task for service providers to achieve. This is because the optimal amount of such resources is a function of various parameters, such as the desired/agreed quality of service (QoS), the service characteristics/profile, workload and service life-cycle. The advent of frameworks that foresee the dynamic establishment and placement of service and network functions further contributes to a decrease in the effectiveness of traditional resource allocation methods. In this work, we address this problem by developing a mechanism which first performs service profiling and then a prediction of the resources that would lead to the desired QoS for each newly deployed service. The main elements of our approach are as follows: (a) the collection of data from all three layers of the deployed infrastructure (hardware, virtual and service), instead of a single layer of the deployed infrastructure, to provide a clearer picture on the potential system break points, (b) the study of well-known container based implementations following that microservice paradigm and (c) the use of a data analysis routine that employs a set of machine learning algorithms and performs accurate predictions of the required resources for any future service requests. We investigate the performance of the proposed framework using our open-source implementation to examine the case of a Hadoop cluster. The results show that running a small number of tests is adequate to assess the main system break points and at the same time to attain accurate resource predictions for any future request.

**Keywords:** heterogeneous monitoring; machine learning; next generation networking; software defined networking; hadoop

check for updates

**Citation:** Uzunidis, D.; Karkazis, P.; Roussou, C.; Patrikakis, C.; Leligou, H.C. Intelligent Performance Prediction: The Use Case of a Hadoop Cluster. *Electronics* **2021**, *10*, 2690. https://doi.org/10.3390/electronics10212690

Academic Editor: Agapito Ledezma Espino

Received: 15 September 2021
Accepted: 28 October 2021
Published: 3 November 2021

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Copyright:** © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

## 1. Introduction

In the context of Network Function Virtualization (NFV), a Network Service (NS), (e.g., router, firewall, cache server etc.), consists of a chain of interconnected Virtual Network Functions (VNF). These virtual functions can physically reside on either single or multiple VNF Infrastructures (NFVI), providing the network operators with significant configuration flexibility. Considering the dynamic nature of the NSs, their life cycle management is not a straightforward procedure. Either because it runs for a short period of time just to provide a specific service, or as the user's demands may change, the NS needs to be adapted to the new requirements by performing dynamic scale up/down actions. To address the challenges that occur in this highly dynamic environment, ETSI has already defined automated mechanisms for rule-based automatic scaling and healing mechanisms in the NFV-Management and Orchestrator (MANO) reference architecture [1]. However, dynamic and optimal resource allocation remains a very interesting research area as its dynamic management cannot rely solely on simplistic rule-based approaches,

making it is necessary to adopt more sophisticated schemes including Artificial Intelligence (AI)/Machine Learning (ML) algorithms. One of the main challenges for a network operator is to maximize the number of supported user services while ensuring the pre-defined QoS for the duration of the whole NS lifecycle. Another key parameter that is of great interest for the providers is the Quality of Experience (QoE) which constitutes, in essence, the quality that the user experiences and is primarily affected by the perceived latency measured by the user to the cloud [2]. In fact, optimal resource allocation is an issue that concerns the Service Providers (SP) at the beginning of the cloud infrastructures, and as the visualization technologies expand from Virtual Machines (VM) to containers and microservices, the optimization problem has become even more complex.

Traditionally, to address this challenge, the network engineers allocate a significantly higher number of resources than required, which directly leads to a waste of resources, as the resources are underutilized. In order to avoid this waste, it is desirable for the network engineer to allocate the exact number of required resources, leading to, ideally, zero underutilization. To achieve this aim, an accurate prediction of the resources required across the whole NS lifecycle is necessary. In particular, this prediction will define the precise point at which the number of resources allocated ensures both a compliance with the agreed Service Level Agreement (SLA) and zero underutilization, which, on one hand, will avoid resource over-allocation and on the other hand will avoid the under-allocation of resources which may lead to a violation of the SLA.

The definition of this "critical point" is a complex problem as it requires (a) an extensive NS profiling for a large number of configurations, (b) the monitoring of a vast amount of system metrics from different sources, such as bare metal, virtual machines, containers, services, networking, etc., and (c) an integrated platform which can perform data analysis for all the monitored metrics and can extract the most significant metrics that can potentially lead to a QoS degradation. The issue of "critical point" determination is becoming more challenging as new types of services emerge every day and the decisions for their resource profile and assignment have to be performed faster and at a more precise level. Moreover, another significant challenge is the heterogeneity of the infrastructure used in modern datacenters. For example, hardware equipment (i.e., physical servers, switches, routers, etc.) and virtualization technologies based on VMs and containers, (i.e., VMware ESXi, Openstack, AWS, linux containers, docker swarm, Kubernetes etc.) are combined to provide an infrastructure of computational and networking resources. As a consequence, SPs require a generalized solution which can monitor, analyze and manage the allocated resources at various levels of the adopted method of implementation.

In this complex environment, ML can effectively aid network engineers to minimize the distance from the critical point of operation. Furthermore, it can automatically extract the profiles for new services as they emerge, exploiting only a small number of deployed configurations thereby minimizing the overall profiling time. These profiles are then used to predict the exact amount of resources required to ensure the QoS for the duration of the whole service life-cycle. As a consequence, an automated process of service profiling and resource performance prediction can provide an improved utilization of the infrastructural resources without the need to sacrifice QoS and/or service availability. The service profiling and accurate performance prediction is a problem which extends to various service categories, such as networking, security, big data, storage, emulation, etc., and the solution we propose here can be easily adapted to any service category as it is based on open-source technology and it is modular, while it shows a great deployment flexibility.

In particular, in this work, we design and deploy a novel framework to address the problem of service profiling and to predict the system "critical points", focusing on complex services running over containers. The proposed solution is a multi-layer approach, as it monitors and analyses data from all three layers of implementation, namely the physical, virtual and service layers. Moreover, it employs open-source technology, while it supports several types of virtualization technologies, providing a high level of flexibility. In addition, it is able to profile various different types of services by performing "white box" testing by

gathering performance metrics directly from the under-test services, or "black box" testing using well-known benchmarking tools (i.e., jMeter, Apache benchmark, etc.). During the data analysis, the proposed implementation exploits the broad gamut of ML algorithms, spanning from polynomial regression to deep neural networks, tailoring the most accurate algorithm for the service under examination. Finally, the candidate architecture is able to designate the most obvious critical points during the service profiling phase, while during the data analysis phase it can extract the hidden critical points. This is possible as it can perform predictions for configurations which were not possible to examine during the profiling phase.

To examine the feasibility of the proposed approach, we created a sand box environment in which we deployed a Hadoop cluster in order to perform service profiling and performance predictions by monitoring an extensive number of critical system metrics (e.g., CPU usage, memory usage, service throughput etc.) from three layers, namely the physical, virtual and service layers. In this work, the specified layers are not related to the OSI-compliant layers. The Hadoop was selected as the tested service since it combines some very desirable features, such as (a) being a widely adopted solution for big data problems, (b) it is open-source, (c) its easy integration with existing infrastructures, and (d) it provides a significant number of test benchmarks which can be exploited to calculate the performance of the deployed infrastructure.

The rest of the paper is organized as follows: in Section 2 we analyze the relevant research in this field in order to provide a holistic view of the problem we address. In Section 3 we introduce the three-layer architecture and in Section 4 we first analyze the proposed test-bed which is deployed to test the feasibility of our solution, and we also discuss the research findings from the conducted tests. Finally, Section 5 concludes the paper and proposes potential future directions.

## 2. Related Work

In prior studies, several different implementations have been introduced to profile and perform predictions on critical system metrics. In [3–6], the profile was performed in test environments which are deployed on the cloud, while in [7–18] implementations in the context of NFV are examined, because they focus on the deployment and validation of VNFs. Most notably, [4] exploits various ML methods to perform predictions using only a subset of the possible configurations, testing TeraSort, PageRank and Single Source Shortest Path (SSSP), attaining a high level of accuracy despite using only a limited number of configurations. Next, [5] uses various algorithms, such as active learning to analyze the performance of the employed framework while [6] exploits a decision tree to sample the deployment space and improve the overall prediction accuracy. In particular, in [6] each deployment configuration represents a set of parameters which directly impact the overall application performance. For example, in the Hadoop Wordcount, the target metric is the execution time while the deployment space comprises the YARN nodes, the number of cores per node, the memory per node and the dataset size. An important limitation of the implementations in [3–5] is that they exploit data solely from the application layer while the performance metrics of the hardware and the virtualization layer remain unexplored. It is worth mentioning that the mechanism presented in [6] can monitor data from physical and application layers, albeit the virtualization layer remains unutilized. This is a significant drawback, because an approach which bypasses the impact of the different application parameters on a specific layer, e.g., the virtualization or physical layers, (a) does not provide knowledge on the relationship between these parameters and the performance metrics and (b) may neglect additional system limiting points which can be attributed to the configurations of the virtualization or physical layers.

Next, the authors in [7] designed a framework to characterize VNFs and highlight their potential bottlenecks under different operational conditions while in [8,11] an entire service chain, which comprises a number of VNFs, has been analyzed, rather than isolated VNFs. The performance profiling from a service chain perspective is highly advantageous

compared with the case of profiling each VNF separately, as it can significantly improve the overall modeling accuracy. Further, [9] proposes a method with monitoring and debugging functionalities which can obtain results in a very short time frame, e.g., seconds to minutes, allowing for rapid reboots and reconfigurations of the examined VNFs or the emulator network. Moreover, [10] introduces a framework which can perform benchmarking on the NVF environment in an automated way and for a different number of operating conditions. The works [7–11] are accurate and efficient methods for service monitoring and profiling, but they do not perform predictions of QoS for different deployment options. This can make the resource allocation mechanism cumbersome, since an accurate and efficient method which can estimate the resources that need to be assigned for a specific NS does not exist, even for configurations which have not been examined during the profiling phase.

To resolve this problem, ML has been exploited in the works of [12–18] for analysis and performance predictions as it can relate the workload parameters with the performance metrics. These proposed ML methods are tailored to the investigated service or VNF and span from linear regression to artificial neural networks. From the aforementioned solutions, only [14] collects and analyzes data from all three layers, however, with the sole purpose to provide a comprehensive view behind the reasons of a performance abnormality. Further, the works of [19,20] address the importance of a unified representation infrastructure; however, a comprehensive analysis of the performance features and metrics from multiple layers, their relationship with the service workload parameters and the identification of the potential breaking points in all three layers, remains unexplored.

In the current work, we introduce a three-layer implementation which monitors and analyzes data from all three layers in order to perform service profiling and performance predictions using a number of different ML algorithms, spanning from linear regression to neural networks. The proposed architecture is able to define the obvious system breaking points for all three layers during the profiling phase and the hidden layers during the analysis phase. This is important from an SP perspective, as it can help to avoid the waste of resources through resource utilization and can avoid an SLA violation through resource overutilization. Moreover, our approach can collect and analyze metrics from not only the hypervisor but also from containers (Kubernetes). It can also be integrated easily with existing MANO frameworks while it uses open-source tools to minimize the overall CapEx costs. To further situate our work, we summarize the most important details of the related literature in the context of NFV in Table 1.

**Table 1.** Related work in the context of NFV.

| Ref. | Testing Service/App | Examined Metrics | Examined Layers | ML Algorithms |
|---|---|---|---|---|
| [6] | Spark k-means, Spark Bayes, Hadoop Wordcount, MongoDB | Execution time, throughput | service mainly | Regression Trees |
| [7] | Clearwater, Snort, Suricata | Successful call rate, CPU, memory and network usage, Packet processing speed vs. traffic | virtual | - |
| [8] | single Video Encoder (VE), blackbox profiling scenario, entire service chain | Number of CPU cores, CPU time | virtual | - |

**Table 1.** *Cont.*

| Ref. | Testing Service/App | Examined Metrics | Examined Layers | ML Algorithms |
|---|---|---|---|---|
| [9] | Intrusion Detection System (IDS) | CPU, memory, traffic rate, packet loss | virtual | Rule based |
| [10] | SIPp prober | Avg. transaction rate, CPU usage | virtual | - |
| [11] | A chain of three VNFs | Throughput vs. CPU time | virtual | Plug-in arbitrary analysis scripts |
| [12] | The service Media Gateway (MG) composed of various VNFs | CPU load, network drops, rejected calls, latency | virtual | Stochastic Gradient Descent regressor (SGDR) |
| [13] | MME, S-GW, HSS, PCRF, PDN-GW | Quality of Decisions (QoD) | virtual | Q-Learning approach |
| [14] | 7 (such as Suricata-IPS, NAT, Tcpdump, Firewall, Netsniffer-ng) | Many from each layer, such as CPU, memory usage, Disk read/write I/O requests and bytes etc. | service virtual physical | Various criteria for behavior classification |
| [15] | HSS, MME, PGW-U-SGWU, SGW-C-PGW-C, INET-GW and eNB | CPU, CPU cache, memory, bandwidth, Disk | virtual | Decision tree-based multilabel classification technique |
| [16] | virtual router, switch, firewall and cache server | CPU usage, packet loss, cache response time | virtual | Linear Regression, k-NN, Interpolation, ANN, Curve Fitting |
| [17] | virtual firewall, (pfSense), virtual streaming server (Nginx) | CPU usage, packet loss, lag ratio | virtual | Support Vector Regression, Random Forest, Gaussian Process, k-NN, Interpolation Method, Curve fitting |
| [18] | Squid cache, Nginx proxy | CPU, total delay, # of VNF instances | virtual | Linear regression, support vector regression, decision trees, ensemble learning, neural networks |
| [19] | Scalable monitoring framework | NS, VNFs, NFVI, SDN controllers | virtual | Rule based |

**Table 1.** *Cont.*

| Ref. | Testing Service/App | Examined Metrics | Examined Layers | ML Algorithms |
|---|---|---|---|---|
| This work | Hadoop | CPU usage, disk usage, memory usage, throughput, average I/O rate | service virtual physical | Linear Regression, Polynomial Regression, Decision Tree, Random Forest, Support Vector Regression, k-NN, and Rule based |

## 3. Proposed Architecture

Despite the fact that optimal resource allocation has already been explored in the literature, a complete, low cost, solution for collecting and analyzing information from all types of monitoring targets, suitable for integration with existing frameworks, is missing. The proposed framework (Figure 1) combines state-of-the-art monitoring and analysis tools in order to provide a reliable and cost-effective resource allocation framework consisting of the following two main components: (a) monitoring framework and (b) an analysis server. This framework can be used as part of an NFV MANO, acting as an enhanced Policy Manager or it can be used as an external component that analyzes the performance of the running NS and provides recommendations to the MANOs using their APIs. This aspect was important to consider in the design and the selection of the tools, although the actual integration with a specific MANO implementation is not within the scope of this work. The key requirements of the proposed system are as follows:

- An automated collection of monitoring data from as many heterogeneous types of sources as possible.
- A sophisticated analysis mechanism based on many AI/ML algorithms for dynamic resource allocation.
- Its easy integration with existing orchestrator frameworks.
- A scalable and resilient architecture ready to transfer a high number of monitoring entities that supports hierarchical deployments.
- Support of near real-time responsiveness.

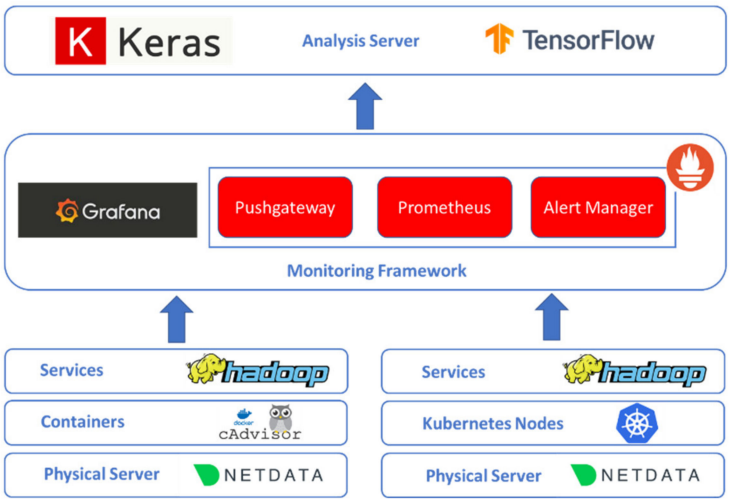



**Figure 1.** Monitoring and analysis architecture.

### *3.1. Monitoring Framework*

The proposed monitoring solution complies with the above requirements by adopting Cloud Native (CN) tools, which can easily integrate new types of monitoring targets without difficult configurations or extensive down-time. Also, all the employed tools are opensource implementations and they are supported by big communities as they are widely used in the industry. These features are crucial because they ensure the reliability of the system and at the same time, they make the integration process with many external systems that use the same tools easier. So, the choice of open-source implementation, which is supported by a large and active community such as the Linux Foundation, Apache, etc., ensures the involvement of many developers for support, bug fixing, and upgrades. Furthermore, as a tool becomes more popular, it is adopted by new systems, which makes integration between systems easier. The monitoring framework consists of monitoring tools for data collection and storage from agents, collecting metrics across different time periods for the targets:

The proposed implementation includes the following monitoring tools:

- The Prometheus server [21] is the central point of event monitoring, storage and alerting. All performance metrics are collected using an 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) the use/support of a flexible query language (PromQL), which makes the interconnection with external systems easier (b) the existence of many opensource implementations (exporters) for exposing the monitoring metrics of various applications, and the ease of creating new ones, (c) the autonomy that is provided as it does not rely on any complex distributed storage mechanisms and (d) the fact that new monitoring targets can be easily added via reconfiguration or using the file-based service discovery mechanisms.
- The Prometheus Pushgateway [22] allows batch jobs and short lived microservices to expose their metrics to Prometheus. Since this function may not exist long enough to be scraped, the metrics can instead be moved to a Pushgateway. The Pushgateway then exposes these metrics to the Prometheus server.
- The Alertmanager [23] handles alerts sent by client applications such as the Prometheus server. It is responsible for deduplicating, grouping, and routing the alerts to the correct receiver integrations such as email, PagerDuty, or OpsGenie. It also attends to the silencing and inhibition of alerts, which is useful for the management of the number of generated notifications.
- Grafana [24] is an open-source solution that obtains those metrics and alerts that are understandable from the Prometheus server, and it provides interactive visualization web dashboards. These dashboards simplify the virtualization of the collected performance metrics so that following each notification the user can refer to a specific dashboard and identify the problem by observing charts.

The proposed implementation includes the following monitoring Agents:

- Netdata.io [25] is a powerful real-time monitoring agent which collects thousands of metrics from systems, hardware, virtual machines, and applications with zero configuration. It runs permanently on the physical/virtual servers, containers, cloud deployments, and edge/IoT devices, and is perfectly safe for installation on a system mid-incident without any preparation.
- cAdvisor [26] provides metrics of the resource usage and performance characteristics of the running containers. It is a running daemon that collects, aggregates, processes, and exports information about running containers. It maintains specific resource isolation parameters for each container, historical resource usage, histograms of complete historical resource usage and network statistics.

It is worth mentioning that some of the most known open-source MANO frameworks in the context of the NFV, such as Service Programming and Orchestration for Virtualized Software Networks (SONATA) [27] and Open Source MANO (OSM) [28] have already

adopted Prometheus as their monitoring server. This makes integration with them straightforward, and only requires updating the configuration file of the Prometheus monitoring server. Moreover, for large scale deployments, Prometheus Monitoring servers support a distributed (cascaded) architecture: the local Prometheus servers collect and store metric data from the different targets, while only the alerts are sent to the federated Prometheus server for further processing and alerts. Another scalability requirement concerns the large flow of data from the monitoring agents to the monitoring server and its respective database, which might affect the service performance in extreme cases. To overcome these potential problems, the monitoring system (a) is configured to store monitoring data of a specific period and, (b) in cases of large deployment, the adoption of the described cascaded architecture provides an appropriate solution.

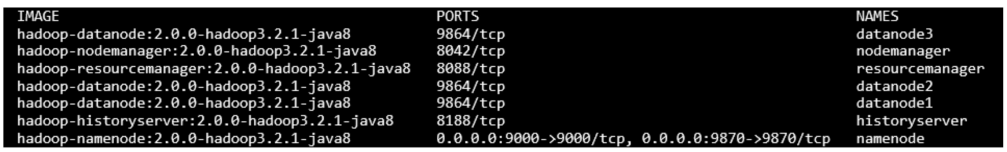
### *3.2. Analysis Server*

The analysis server is provided with data which were collected during the service profiling phase and comprises a broad gamut of ML algorithms, spanning from linear regression to deep neural networks. Depending on the characteristics of the examined service, e.g., (a) the complexity between the workload parameters, such as the file size, the number of files, and the performance metrics, such as the CPU and memory usage, and (b) the number of examined features, as the activated ML algorithms are tailored to the specific service. Their overall modeling accuracy depends both on the dataset (quantity and quality of the measured data) and on the selected ML algorithms. For example, a more complex mathematical relationship between the workload parameters and the performance metrics may require more complex ML structures, e.g., a deep neural network with a larger number of hidden layers. Furthermore, the proposed implementation allows for algorithm re-training with additional data, as the service continues profiling for a longer time period, potentially leading to a higher modelling accuracy. Overall, the methodology for the selection of the ML algorithms is as follows. The predicted values for each ML algorithm are compared against the actual values for each monitored metric, and once the modelling accuracy verifies a pre-defined threshold, the most accurate algorithm is employed from this point onwards.

## 4. Implementation Test Bed—Evaluating a Real-Life Service

### *4.1. The Deployed Sandbox Test-Bed*

In the presented work, specific attention is paid to the evaluation of a complex service consisting of several containers using microservices. One such service is the Apache Hadoop, which is an open-source software framework that offers big data and analytics services with almost unlimited scalability. It is used to store and process large datasets with great efficiency and can be deployed conveniently on commodity hardware. Since its release in 2011, it has been widely used in the industry as either an on-premise or cloud-based solution. Many web service providers offer Hadoop as a service, based on their own deployments or using platform-focused vendors such as Hortonworks or MapR. Nowadays, Hadoop is available through many service providers, namely Cloudera, Hortonworks, MapR, Amazon Elastic Map Reduce, Altiscale, etc. For our experiments, an Apache Hadoop cluster was deployed (Figure 2) consisting of several microservices, namely, historyserver, a namenode, a nodemanager, a resource manager, and three data node servers.



| IMAGE | PORTS | NAMES |
|---|---|---|
| hadoop-datanode:2.0.0-hadoop3.2.1-java8 | 9864/tcp | datanode3 |
| hadoop-nodemanager:2.0.0-hadoop3.2.1-java8 | 8042/tcp | nodemanager |
| hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8 | 8088/tcp | resourcemanager |
| hadoop-datanode:2.0.0-hadoop3.2.1-java8 | 9864/tcp | datanode2 |
| hadoop-datanode:2.0.0-hadoop3.2.1-java8 | 9864/tcp | datanode1 |
| hadoop-historyserver:2.0.0-hadoop3.2.1-java8 | 8188/tcp | historyserver |
| hadoop-namenode:2.0.0-hadoop3.2.1-java8 | 0.0.0.0:9000->9000/tcp, 0.0.0.0:9870->9870/tcp | namenode |

**Figure 2.** Under test Hadoop cluster.

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 specification for the physical servers is summarized in Table 2, as is the networking device that was used is the Cisco SG250 18 port Gigabit switch.

**Table 2.** Physical Servers technical specifications.

| PowerEdge R230 | |
|---|---|
| CPU | 4 CPUs × Intel(R) Xeon(R) (CPU E3-1220 v6 @ 3.00 GHz) |
| RAM | 16 GB DDR4 |
| HDD | 8 TB |
| NETWORK | 2 × 1GbE LOM |

In the first server, we deployed all the components of the proposed architecture using docker-compose scripts, and on the second, following the same approach, we deployed the Hadoop cluster and the monitoring agents (cAdvisor and Netdata.io). All of the software tools (Table 3) 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 redeployable. Furthermore, the adoption of the Prometheus monitoring server prepares the proposed solution to integrate and collect monitoring data from Kubernetes clusters.

**Table 3.** Open-source monitoring and analysis tools.

| Software Tools | Versions |
|---|---|
| Prometheus Server | v 2.19.3 |
| Prometheus Pushgateway | v 1.2.0 |
| Prometheus Alertmanager | v 0.23.0 |
| cAdvisor | v 0.32.0 |
| Netdata.io | v 1.23.2 |
| Apache Spark | v 3.2.1 |

### *4.2. The Evaluation Procedure*

The "testDFSIO" is a well-known read and write test for the Hadoop Distributed File System (HDFS). In particular, it is used to test the performance of NameNode and network components in HDFS. This test was executed using a MapReduce job, which first splits the input datasets into independent sets of data to be processed in parallel (map part), after which the outputs of the map are combined into a smaller set of values (reduce part). We selected MapReduce as it is a widely adopted programming model that provides significant advantages when there is a large amount of data needed to be processed. In particular, it is simple and scalable as it can process large amounts of data in a reasonable timeframe, and it can handle multiple sources of data and multiple types of data. Furthermore, the user is able to select both the number of files and the file size for each test. In addition, the user can set the type of test (read or write) and when the test is completed, specific service layer metrics can be visualized, such as the throughput, the average input-output (IO) rate, the standard deviation of the IO rate and the execution time of the test. Overall, the "testDFSIO" is an important test for the Hadoop cluster as it provides an overall performance evaluation of the provided service and a fast impression of how efficient the cluster is in terms of IO.

In this work, we exploit this test in order to emphasize the deployed implementation while monitoring data from all three layers, particularly from the "testDFSIO" (service layer), hardware and virtual layers. Next, the collected data were exploited to perform ML predictions. For modelling purposes, we selected six ML algorithms to mathematically

relate the input parameters, namely the number of files and file size, to the monitored metrics, e.g., CPU and disk usage. The employed ML algorithms are: Multiple Linear Regression, Multivariate Polynomial Regression, Decision Tree, Random Forest, Support Vector Regression and k-Nearest Neighbors. The list of ML algorithms available in the analysis server of the proposed implementation is significantly larger, containing even more complex algorithms, e.g., deep neural networks (DNN). However, in the current analysis, we excluded DNN as it requires a large amount of training data in order to provide accurate results. As this evaluation procedure has a small number of training data (55 cases are considered), the DNN will definitely provide erroneous estimations. In other service deployments, where the amount of training dataset is significantly higher, the exploitation of deep learning algorithms must be considered. The six algorithms previously mentioned were selected as they provide high modeling accuracy within a relatively low computational timeframe and are also the most suitable algorithms for problems containing a small number of features (in our case, two). In our implementation, we utilize Python3 as the programming language, and 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 Tensorflow library (version 2.0.0)as the backend.

### *4.3. Results*

In this section, we first examine the data collected from all three layers during the service profiling phase (Section 4.3.1) and we then import these data into the six ML algorithms in order to perform predictions about the selected metrics (Section 4.3.2).

#### 4.3.1. Profiling of Critical System Metrics from Three Layers

The configured service is stressed using the "testDFSIO" test in order to examine the breaking points of our implementation, or in other words, the parameters within which the proposed service can operate seamlessly. A thorough understanding of these limits is very important from an SP perspective, as it can ensure the QoS during the overall service lifecycle. In our tests, we created a write test for two of the "testDFSIO" features, namely the number of files and the size of each file, in order to stress the implemented HDFS. Further, to obtain meaningful results, we ensured the same initial conditions for each test, by deleting all the files which were written during the previous test from the disk.

We profiled the metrics for the three layers, and we illustrate the six most representative metrics in Figure 3 (two from each layer). As shown in Figure 3, both the CPU usage (Figure 3a) and disk usage (Figure 3b) metrics increase as the number of files scale, since a larger number of system resources need to be allocated to complete the test. Next, to select the metrics that we will take into consideration from the *virtual layer*, we chose to focus on the performance metrics that refer to the node manager container (among the different nodes of the Hadoop implementation), because the node manager is the most resource consuming node and thus plays a critical role in this test. As is evident, the memory usage of the node manager (Figure 3d) increases when the number of files scales and when the file size increases. Furthermore, the impact of the CPU usage (Figure 3c) strongly depends on the file size, for example, in cases of a 100 MB file size and 25 files and above, the impact of the CPU usage is expected to reach 200%. The value of the CPU usage exceeds 100%, which is attributed to the fact that we use a multi-core physical machine, and also to the fact that the test is implemented as a multi-thread procedure. Next, from the *service layer* we selected the throughput (Figure 3e) and the average I/O rate (Figure 3f), both of which are provided from the write "testDFSIO" test. The results reveal that (a) when the number of files is smaller than five, both the throughput and the average I/O rate decrease as the file size increases; (b) when the number of files is five and above, both metrics reach a plateau; more specifically, when the file size is 500 MB and above, their values are minimized, constituting an additional system limiting point.

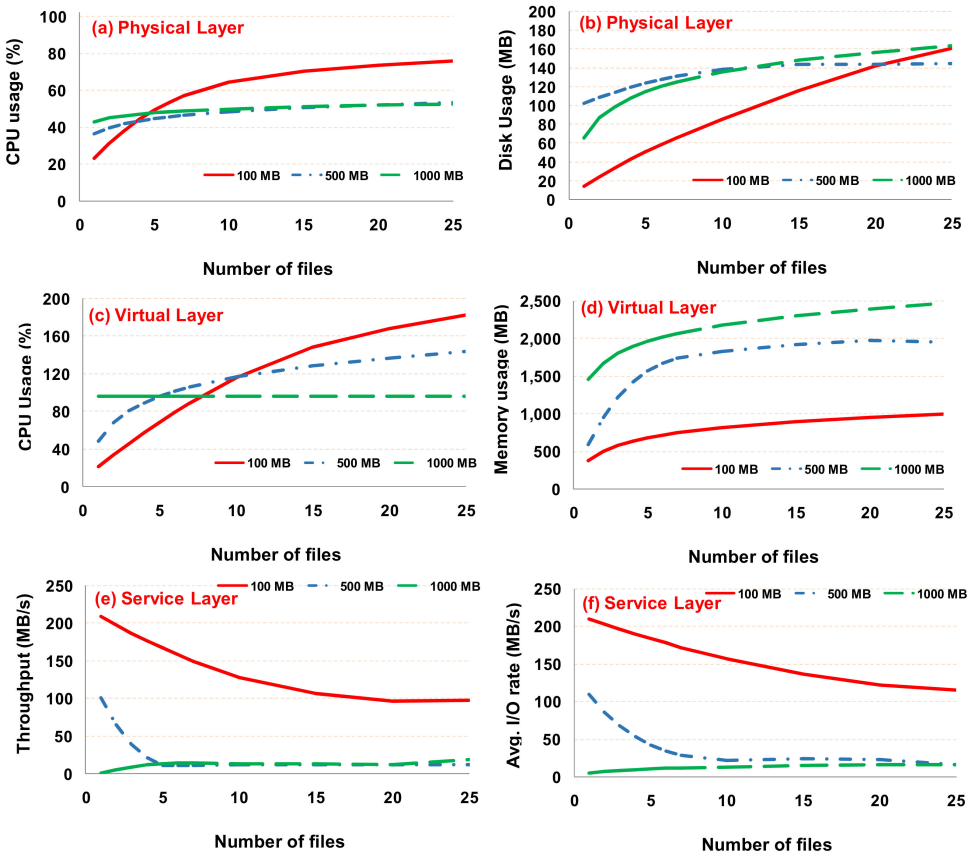



**Figure 3.** Critical system metrics monitored from three layers. (**a**,**b**) physical layer metrics, (**c**,**d**) virtual layer metrics, (**e**,**f**) service layer metrics.

The aforementioned results confirm the need to monitor and analyze data from multiple system layers, namely the physical, virtual and service layers in order to identify all the potential system limiting points and to identify the exact conditions that are of detriment to the overall performance of the implementation.

Next, we measure the overall execution time of each test for a different number of files and file sizes (Figure 4). This metric is provided by the write "testDFSIO" test after the completion of each test. We can observe that the execution time of each test increases almost linearly with an increase in the number of files. This is expected, as the overall implementation requires a greater amount of time to complete the write test due to the larger size, which is required to be written in the disk. The execution time may exceed 10 min in some tests, showing that the deployed sandbox system has reached its limits, as after this value, e.g., after 30 files, the allocated disk space is fully populated with written data.

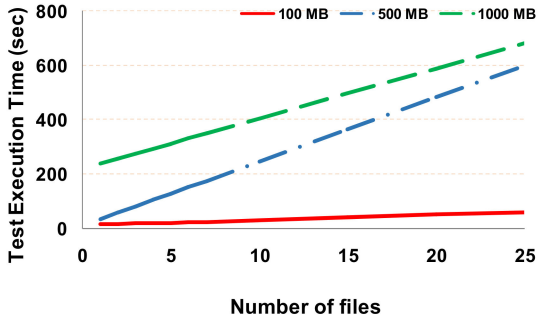



**Figure 4.** Execution time of testDFSIO for the different examined cases.

4.3.2. Predictions Using Machine Learning

In this section, we present the results of the ML analysis on the six metrics presented in Figure 3 using six well-known ML algorithms, namely the Multiple Linear Regression (LR), Multivariate Polynomial Regression (PR), Decision Tree (DT), Random Forest (RF), Support Vector Regression (SVR) and the k-Nearest Neighbors (k-NN) algorithm. The data analysis is highly advantageous for an SP as it can (a) predict the behavior 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, e.g., the CPU usage on the node manager. There were 55 data points which were measured using the write test "testDFSIO" for each metric and they were used to feed the ML algorithms. A representation of the relationship between the input parameters (number of files and file size) and the output parameters (performance metrics, e.g., CPU usage) for indicative cases is illustrated in Figure 3. The number of data points is relatively small as there are a small number of features in this specific problem (two), which are the number of files and the file size. We decided to consider a limited amount of data points (55), to identify the ML algorithms that would lead to very high modeling accuracy with a limited amount of data so that we achieve fast service profiling. The data points were attained by iterating over the following parameters:

Number of files: [1, 2, 3, 4, 5, 6, 7, 10, 15, 20, 25]

File size (MB): [100, 250, 500, 750, 1000]

The ML task was performed via the following steps. In the first step, the 55 collected pairs of input (number of files and file size) and output values for each target metric (e.g., CPU usage) were first shuffled and then divided into three sub-sets, one for training which included 60% of the available points, one for validation which incorporated 20% of the available points/values and one for testing which constituted of the final 20% of the collected points/values. Next, a circular rotation between the three sub-sets was performed in order to ensure that all the collected values for each metric were tested. In the second step, the training sub-set was used to train the ML algorithm while the optimal hyperparameters were derived from the validation sub-set. 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 values 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:

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

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

Table 4 provides the optimal hyperparameter values for the six examined metrics, which were obtained using a grid search for the six ML algorithms. As is evident from Table 4, the linear regression exploits a first order polynomial. For a higher order polynomial, we select a second order, because a higher order polynomial, e.g., of a third order, would lead to overfitting in the training data. Furthermore, when the DT is employed, a tree with a depth equal to five is adequate. Next, the number of estimators of RF spans between four and five and in the case of SVR, the parameter $\gamma$ ranges from $5 \times 10^{-5}$ and 0.01. Finally, when the k-NN is employed, it is adequate to calculate the distance from only one or two of the closest neighbors. In general, the value of each hyperparameter can impact the algorithmic architecture and its elaborateness, e.g., a greater polynomial order leads to a more complex structure. In summary, our analysis reveals that our problem does not require ML structures that are overly complex.

**Table 4.** Optimal hyperparameters for each ML algorithm and metric.

| Method | Hyperparameter | Optimal Hyperparameter Value/Metric | | | | | |
|---|---|---|---|---|---|---|---|
| - | - | CPU Usage PM | Disk Usage PM | CPU Usage NM | Memory Usage NM | Throughput | Average I/O rate |
| Multiple Linear Regression (LR) | order of polynomial | 1st | 1st | 1st | 1st | 1st | 1st |
| Multivariate Polynomial Regression (PR) | order of polynomial | 2nd | 2nd | 2nd | 2nd | 2nd | 2nd |
| Decision Tree (DT) | depth | 5 | 5 | 5 | 5 | 5 | 5 |
| Random Forest (RF) | depth, number of estimators | 5, 4 | 5, 4 | 5, 4 | 5, 5 | 5, 5 | 5, 5 |
| Support Vector Regression (SVR) | C, $\gamma$ | 50, 0.01 | 50, 0.0075 | 50, 0.01 | 50, $5 \times 10^{-5}$ | 50, $5 \times 10^{-3}$ | 50, $5 \times 10^{-3}$ |
| k-Nearest Neighbors (k-NN) | k | 2 | 2 | 2 | 2 | 1 | 1 |

The results of the applied ML algorithms on the six metrics are illustrated in Figure 5. The LR shows the highest error in five out of six metrics, which is expected, as in Figure 3 it is shown that the relationship between the two features and the analyzed metrics is not, in most cases, linear. Remarkably, the approximation error exceeds 100% in the service layer metrics, as the predicted value can be greater than two times the actual value and as consequence the fraction $\frac{|y_i - \hat{y}_i|}{y_i}$ in Equation (1) can be greater than "one" for a significant number of the predicted cases. Next, PR is an adequate method for four out of six metrics, providing an error of less than 16% in the physical and virtual layer metrics. On the other hand, the service layer metrics are the most difficult to predict. This can be attributed to the fact that the relationship between the two features and these metrics is more complex, and, in order to improve the estimation accuracy, a greater number of tests must conducted, or a data augmentation method has to be considered. The RF slightly outperforms DT by 0.3% to 4.5%, which is expected, since it incorporates multiple trees. Furthermore, the accuracy of the SVR depends considerably on the studied metric and, in most cases, it manages to attain an error of less than 30%. Finally, k-NN attains the highest accuracy in the service layer metrics, which are the throughput and average I/O rate while PR, RF and k-NN are the most accurate metrics for both physical and virtual layer metrics, which are the CPU and memory usage.

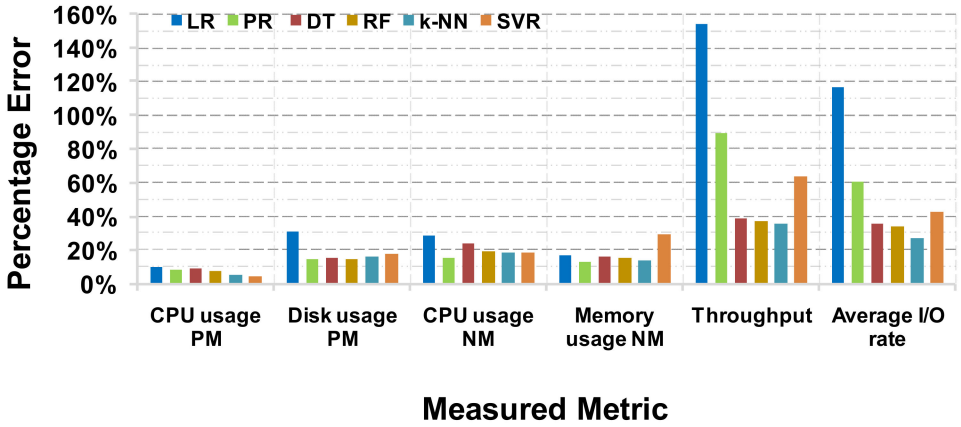



**Figure 5.** Comparison of six well-known ML algorithms applied to six system metrics.

This study confirms that ML algorithms can accurately predict the performance of critical system metrics, using only a minimal fraction of different system conditions. These are promising results, which can be further improved upon with data augmentation techniques and/or a dataset enrichment with a greater number of tested cases, especially for the service layer metrics.

## 5. Conclusions and Future Work

We proposed a three-layer implementation which can perform service profiling and predict the most important system metrics of the examined implementation for any possible operational scenario. We demonstrated that by using only a small number of data points we can attain a sufficient accuracy even when the employed ML algorithms are loaded with a very limited dataset. These results can be used to minimize the distance of a system's operation from its "critical point" leading to significant resource savings while ensuring the QoS. The proposed solution is based on open-source tools, and it is also scalable as it allows for the collection of metrics for various components. Finally, it is flexible as it can easily host different services as it employs a different type of benchmarking tools, such as testDFSIO, jMeter, and Apache benchmark.

In future, we plan to perform data augmentation which can directly lead to a higher modelling accuracy while reducing the need for additional and time-consuming tests on the deployed infrastructure. Moreover, we plan to test our implementation with services in other representative domains such as in media applications, security etc. Furthermore, we intend to perform online machine learning training for each new NS, as well as to integrate the proposed architecture with some of the open-source MANO frameworks in order to provide a completely autonomous management system ready to be used in the NFV context. Another research avenue to pursue is whether this architecture could be deployed at the network edge, which would allow for a better utilization of resources and thus permit Artificial intelligence applications to be executed closer to the devices (things) where the data are generated. This would significantly bolster the implementation of the edge, fox and cloud continuum that is currently being widely researched.

**Author Contributions:** Conceptualization, P.K.; methodology, D.U. and P.K.; software, P.K. and C.R.; formal analysis, investigation, D.U. and H.C.L.; resources, H.C.L.; data curation, D.U. and C.R.; writing—original draft preparation, D.U., P.K., H.C.L. and C.P.; writing—review and editing, D.U., P.K., H.C.L. and C.P. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research received no external funding.

**Acknowledgments:** This work is partially supported by University of West Attica.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript

| | |
|---|---|
| AWS | Amazon Web services |
| AI | Artificial Intelligence |
| CN | Cloud Native |
| DT | Decision Tree |
| DNN | Deep Neural Network |
| HDFS | Hadoop Distributed File System |
| IO | Input-Output |
| IDS | Intrusion Detection System |
| k-NN | k-Nearest Neighbors |
| ML | Machine Learning |
| MANO | Management and Orchestrator |
| MG | Media Gateway |
| LR | Multiple Linear Regression |
| PR | Multivariate Polynomial Regression |
| NFV | Network Function Virtualization |

| | |
|---|---|
| NS | Network Service |
| OSM | Open Source MANO |
| QoD | Quality of Decisions |
| QoE | Quality of Experience |
| QoS | Quality of Service |
| RF | Random Forest |
| SLA | Service Level Agreement |
| SP | Service Providers |
| SSSP | Single Source Shortest Path |
| SVR | Support Vector Regression |
| VE | Video Encoder |
| VM | Virtual Machines |
| VNF | Virtual Network Functions |
| NFVI | VNF Infrastructures |

## References


1. Available online: https://www.etsi.org/deliver/etsi_gr/NFV-IFA/001_099/041/04.01.01_60/gr_NFV-IFA041v040101p.pdf (accessed on 23 August 2021).
2. Palumbo, F.; Aceto, G.; Botta, A.; Ciuonzo, D.; Persico, V.; Pescapé, A. Characterization and analysis of cloud-to-user latency: The case of Azure and AWS. *Comput. Netw.* **2020**, *184*, 107693. [CrossRef]
3. Wood, T.; Cherkasova, L.; Ozonat, K.; Shenoy, P. Profiling and Modeling Resource Usage of Virtualized Applications. In Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Leuven, Belgium, 1–5 December 2008.
4. Giannakopoulos, I.; Tsoumakos, D.; Papailiou, N.; Koziris, N. PANIC: Modeling Application Performance over Virtualized Resources. In Proceedings of the 2015 IEEE International Conference on Cloud Engineering, Tempe, AZ, USA, 9–13 March 2015; pp. 213–218.
5. Duplyakin, D.; Brown, J.; Ricci, R. Active Learning in Performance Analysis. In Proceedings of the 2016 IEEE International Conference on Cluster Computing (CLUSTER), Taipei, Taiwan, 12–16 September 2016.
6. Giannakopoulos, I.; Tsoumakos, D.; Koziris, N. A decision tree based approach towards adaptive modeling of big data applications. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017.
7. Cao, L.; Sharma, P.; Fahmy, S.; Saxena, V. NFV-VITAL: A framework for characterizing the performance of virtual network functions. In Proceedings of the 2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN), San Francisco, CA, USA, 18–21 November 2015; pp. 93–99.
8. Peuster, M.; Karl, H. Understand Your Chains: Towards Performance Profile-Based Network Service Management. In Proceedings of the 2016 Fifth European Workshop on Software-Defined Networks (EWSDN), Den Haag, The Netherlands, 10–11 October 2016.
9. Rossem, S.V.; Tavernier, W.; Peuster, M.; Colle, D.; Pickavet, M.; Demeester, P. Monitoring and debugging using an SDK for NFV-powered telecom applications. In Proceedings of the 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Palo Alto, CA, USA, 7–10 November 2016.
10. Rosa, R.V.; Bertoldo, C.; Rothenberg, C.E. Take Your VNF to the Gym: A Testing Framework for Automated NFV Performance Benchmarking. *IEEE Commun. Mag.* **2017**, *55*, 110–117. [CrossRef]
11. Peuster, M.; Karl, H. Profile your chains, not functions: Automated network service profiling in DevOps environments. In Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Berlin, Germany, 6–8 November 2017.
12. Iglesias, J.O.; Aroca, J.A.; Hilt, V.; Lugones, D. Orca: An orchestration automata for configuring VNFS. In Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Las Vegas, NV, USA, 11–15 December 2017; pp. 81–94.
13. Sciancalepore, V.; Yousaf, F.Z.; Costa-Perez, X. z-TORCH: An Automated NFV Orchestration and Monitoring Solution. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1292–1306. [CrossRef]
14. 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), Los Angeles, CA, USA, 28–29 March 2018.
15. Khan, M.G.; Bastani, S.; Taheri, J.; Kassler, A.; Deng, S. NFV-Inspector: A Systematic Approach to Profile and Analyze Virtual Network Functions. In Proceedings of the 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), Tokyo, Japan, 22–24 October 2018.
16. Van Rossem, S.; Tavernier, W.; Colle, D.; Pickavet, M.; Demeester, P. Profile-Based Resource Allocation for Virtualized Network Functions. *IEEE Trans. Netw. Serv. Manag.* **2019**, *16*, 1374–1388. [CrossRef]
17. Van Rossem, S.; Tavernier, W.; Colle, D.; Pickavet, M.; Demeester, P. Optimized Sampling Strategies to Model the Performance of Virtualized Network Functions. *J. Netw. Syst. Manag.* **2020**, *28*, 1482–1521. [CrossRef]
18. Schneider, S.; Satheeschandran, N.P.; Peuster, M.; Karl, H. Machine Learning for Dynamic Resource Allocation in Network Function Virtualization. In Proceedings of the 2020 6th IEEE Conference on Network Softwarization (NetSoft), Ghent, Belgium, 29 June–3 July 2020.

19. Trakadas, P.; Karkazis, P.; Leligou, H.C.; Zahariadis, T.; Papadakis, A. Scalable monitoring for multiple virtualized infrastructures for 5G services. In Proceedings of the International Symposium on Advances in Software Defined Networking and Network Functions Virtualization, Athens, Greece, 22–26 April 2018.
20. Al-Hazmi, Y.; Gonzalez, J.; Rodriguez-Archilla, P.; Alvarez, F.; Orphanoudakis, T.; Karkazis, P.; Magedanz, T. Unified representation of monitoring information across federated cloud infrastructures. In Proceedings of the IEEE 2014 26th International Teletraffic Congress (ITC), Karlskrona, Sweden, 9–11 September 2014.
21. GitHub—Prometheus/Prometheus: The Prometheus Monitoring System and Time Series Database. Available online: https://github.com/prometheus/prometheus (accessed on 23 August 2021).
22. GitHub—Prometheus/Pushgateway: Push Acceptor for Ephemeral and Batch Jobs. Available online: https://github.com/prometheus/pushgateway (accessed on 23 August 2021).
23. GitHub—Prometheus/Alertmanager: Prometheus Alertmanager. Available online: https://github.com/prometheus/alertmanager (accessed on 23 August 2021).
24. GitHub—Grafana/Grafana. Available online: https://github.com/grafana/grafana (accessed on 23 August 2021).
25. GitHub—Netdata/Netdata: Real-Time Performance Monitoring. Available online: https://github.com/netdata/netdata (accessed on 23 August 2021).
26. GitHub—Google/Cadvisor. Available online: https://github.com/google/cadvisor (accessed on 23 August 2021).
27. GitHub—Sonata-nfv. Available online: https://github.com/sonata-nfv (accessed on 23 August 2021).
28. "OSM ETSI" Git. Available online: https://osm.etsi.org/gitweb/ (accessed on 23 August 2021).