The Journal of Systems and Software 136 (2018) 19–38

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review

Salman Taherizadeh[a], Andrew C. Jones[b], Ian Taylor[b], Zhiming Zhao[c], Vlado Stankovski[a,*]

[a] University of Ljubljana, Slovenia
[b] Cardiff University, UK
[c] University of Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 21 May 2017
Revised 22 September 2017
Accepted 31 October 2017
Available online 2 November 2017

Keywords:
Edge computing
Cloud computing
Monitoring technologies
Requirement analysis
Adaptive applications
Taxonomy of monitoring requirements

ABSTRACT

Recently, a promising trend has evolved from previous centralized computation to decentralized edge computing in the proximity of end-users to provide cloud applications. To ensure the Quality of Service (QoS) of such applications and Quality of Experience (QoE) for the end-users, it is necessary to employ a comprehensive monitoring approach. Requirement analysis is a key software engineering task in the whole lifecycle of applications; however, the requirements for monitoring systems within edge computing scenarios are not yet fully established. The goal of the present survey study is therefore threefold: to identify the main challenges in the field of monitoring edge computing applications that are as yet not fully solved; to present a new taxonomy of monitoring requirements for adaptive applications orchestrated upon edge computing frameworks; and to discuss and compare the use of widely-used cloud monitoring technologies to assure the performance of these applications. Our analysis shows that none of existing widely-used cloud monitoring tools yet provides an integrated monitoring solution within edge computing frameworks. Moreover, some monitoring requirements have not been thoroughly met by any of them.

1. Introduction

In recent years, a wide variety of software solutions, such as Internet of Things (IoT) applications, have emerged as cloud-based systems. As a consequence, billions of users or devices get connected to applications on the Internet, which results in trillions of gigabytes of data being generated and processed in cloud datacenters. However, the burden of this large data volume, generated by end-users or devices, and transferred toward centralized cloud datacenters, leads to inefficient utilization of communication bandwidth and computing resources. Since all the resource capacity and computational intelligence required for data processing reside principally in the cloud-centric datacenters, data analytics for current cloud solutions (e.g. for Amazon AWS IoT[1] and Google Cloud Dataflow[2]) is still an open research problem.

To overcome above problem, modern cloud frameworks such as edge (Shi and Dustdar, 2016), fog (Bonomi et al., 2014) and osmotic (Villari et al., 2016) computing are aimed at increasing capabilities and responsibilities of resources at the edge of the network compared to traditional centralized cloud architectures by not only placing services in the proximity of end-users or devices, but also using new data transfer protocols to improve the interaction with datacenter-based services. This also provides a low-latency response time for the application.

Along these lines, the main goal of the SWITCH project[3] is to introduce a novel software design model by which QoS/QoE objectives can be included in the complete lifecycle of applications running in modern cloud frameworks, including edge computing. Based on this conceptual model, SWITCH solution provides an environment for developing software systems and monitoring their execution, and an autonomous platform for adapting system behavior. The focus of the present paper is on monitoring such self-adaptive applications within an edge computing context, where some data processing takes place at the edge of the network, and therefore monitoring and self-adaptation of this processing is nec-

* Corresponding author.
E-mail addresses: Salman.Taherizadeh@fgg.uni-lj.si (S. Taherizadeh), JonesAC@cardiff.ac.uk (A.C. Jones), TaylorIJ1@cardiff.ac.uk (I. Taylor), Z.Zhao@uva.nl (Z. Zhao), Vlado.Stankovski@fgg.uni-lj.si (V. Stankovski).

[1] Amazon AWS IoT, https://aws.amazon.com/iot/.

[2] Google Cloud Dataflow, https://cloud.google.com/dataflow/.

[3] The SWITCH project, http://www.switchproject.eu/.

https://doi.org/10.1016/j.jss.2017.10.033

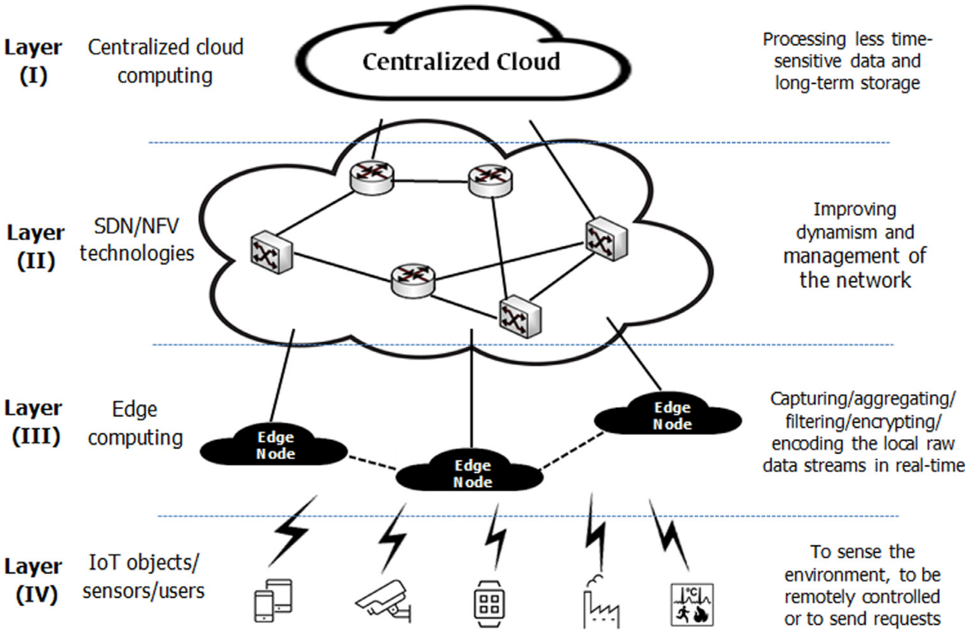

Layer (I)
Centralized cloud computing
Centralized Cloud
Processing less time-sensitive data and long-term storage
Layer (II)
SDN/NFV technologies
Improving dynamism and management of the network
Layer (III)
Edge computing
Edge Node
Edge Node
Edge Node
Capturing/aggregating/filtering/encrypting/encoding the local raw data streams in real-time
Layer (IV)
IoT objects/sensors/users
To sense the environment, to be remotely controlled or to send requests

Fig. 1. An example for multi-layer edge computing frameworks.

essary in order to ensure that effective use is made of centralized cloud facilities, edge devices, and the entire infrastructure that an edge computing framework provides.

As a motivating example, Fig. 1 provides one modern computing framework schema – edge computing architecture – which includes different layers: (I) Centralized cloud computing, (II) SDN/NFV technologies, (III) Edge computing, and (IV) IoT objects, sensors or users.

The multi-layer architecture, shown in Fig. 1 as an example of pioneer cloud computing frameworks, is described below:

Layer (I) Centralized cloud computing layer: The centralized cloud computing layer includes cloud datacenters which can belong to various providers. The centralized cloud computing layer can be utilized for the purpose of long-term storage and application-level data processing operations that are typically less time-sensitive. In other words, the capability of this layer would be using centralized cloud infrastructure in order to run big data analytics and process less time-sensitive data. In this layer, applications may be composed of different, modular services, each one performing different high-level data processing according to users' requirements for various purposes. For instance, an IoT disaster early warning application might have two services in this layer: "warning trigger" service and "call operator" service. A "warning trigger" may be a surveillance service that processes the incoming data measured by sensors in order to notify another service, e.g. "call operator", when irregular incidents occur. The "call operator" service may decide whether or not to send an alert to emergency systems or to the public entities.

Layer (II) SDN/NFV technologies layer: Software-Defined Networking (SDN) (Astuto et al., 2014) and Network Functions Virtualization (NFV) (Jesus-Gil and Botero, 2016) are emerging as new ways of designing, building and operating networks. These two complementary technologies are able to support transition of data between edge nodes and cloud datacenters. Introduction of these technologies can easily improve dynamism and management of the network. For example, it is possible to change the path through which data flows between the centralized cloud computing layer (the first layer) and the edge computing layer (the third layer) if the current network quality is not satisfactory. In combination, SDN and NFV have the potential to specify network flows, enhance network performance and also provide network management abstraction independent of underlying networking devices. In this way, SDN and NFV can be taken into consideration as enabling solutions to steer the evolution of not only network environments, but also new cloud-based application architectures such as edge computing frameworks.

Layer (III) Edge computing layer: At the edge computing layer, edge nodes represent gateways and data capturing services able to act on raw data, for example aggregating, filtering, encrypting and encoding the local data streams in real-time. This layer is a place where the cloud resources are being distributed and moved near to the end-users and end-devices. This is the main reason that edge computing has been called ubiquitous computing as well. The rationale of employing edge nodes is to analyze time-sensitive data closer to the location where these data streams are collected, hence taking some of the computational load off the resources at the centralized cloud computing layer (the first layer) and, in some cases, reducing network load too (Satyanarayanan, 2017) at the SDN/NFV technologies layer (the second layer). Edge nodes can also have other functionalities, dependent on the requirements of individual use cases. In an early warning system, these nodes consist of services to receive data over direct link from sensors, filter the input data stream, aggregate the measured values and send the data to the "warning trigger" which is another service running at centralized cloud computing layer. Hence, an edge computing layer can offload significant traffic from the core network and datacenters. That is why this new paradigm, as an extension of centralized cloud, provides low latency, location awareness, and optimizes users' experience under QoS requirements for time-critical and even real-time applications.

Layer (IV) IoT objects/sensors/users layer: In this layer, connected devices have a pervasive presence in the Internet. Objects (e.g. smart home modules) can be remotely controlled; sensors are able to measure different parameters (e.g. temperature, barometric pressure, humidity and other environmental variables) and also users are capable of using online software solutions via connected devices such as mobile phones.

The multi-layer edge computing architecture depicted in Fig. 1 offers the following improvements over the classical cloud computing model:

- Reducing the amount of network traffic: Edge nodes at the edge computing layer (the third layer) are able to filter unnecessary data and significantly aggregate only key information that should be streamed to the centralized cloud computing layer (the first layer), and then received, stored and processed.
- Improving the application performance: Data processing locally on edge nodes at the edge computing layer (the third layer) next to end-users or end-devices rather than at the centralized cloud computing layer (the first layer) can be exploited as a solution to shrink latency, reduce response time and hence improve the application QoS.
- Facilitating new approaches of load-balancing: The edge computing paradigm has introduced new functionalities of service migration such as movement of running services between the centralized cloud computing layer (the first layer) and the edge computing layer (the third layer) to support load-balancing on-demand (Sharma et al., 2017). Or it can provide a highly-improved load-balancing behavior by scaling the power of computing locally on edge nodes when compared to traditional, centralized computation.
- Providing the awareness of location, network and context information: In consequence of the edge computing architecture, it is now possible to track end-users' information such as their location, mobility, network condition, behavior and environment in order to efficiently provide customized services. This would ensure end-users' needs and preferences for QoE (as direct measurement of users' satisfaction) are accounted for.
- Minimizing energy consumption: Rapid growth in the number of objects and users connected to the Internet has been always been associated with the demand for maximizing the energy efficiency. Tasks can be offloaded from end-devices to edge nodes that are not far away such as centralized cloud datacenters. This fact helps to reduce the energy consumption at end-devices, centralized computing infrastructures and network points between the edge computing layer and centralized cloud computing layer.

The performance of edge computing applications varies significantly depending on runtime variations in running conditions e.g. the number of arrival requests to be processed, availability of virtualized resources, network connection quality between different application components distributed over the Internet, etc. Therefore, tracking dynamic changes of operational environments is essential in order to identify and remedy any deterioration of system health. To this end, the use of monitoring capabilities in every layer of an edge computing framework helps the cloud-based application provider to recognize where any performance bottlenecks are. Besides this, it allows the system to predict potential issues, and to enhance the application performance to avoid QoE degradation experienced by the user.

Such advanced cloud-based frameworks, providing highly distributed, heterogeneous and even federated environments, can exploit lightweight container-related virtualization technologies (such as CoreOS,[4] Kubernetes,[5] OpenShift Origin[6] and Docker Swarm[7]) for the automatic deployment of different services which communicate with each other through well-defined, efficient mechanisms. Therefore, various monitoring requirements in terms of cloud infrastructure, container virtualization, communication network and application-specific condition are present when using edge computing platforms.

The significance of a monitoring system fully-configured for an application adaptation objective has been discussed in experience studies in various cloud contexts. Such studies present a wide range of monitoring technologies and methodologies of diverse applicability in practice. Various research works have analyzed these monitoring tools and approaches needed for cloud applications so far (Aceto et al., 2012; Aceto et al., 2013; Fatema et al., 2014; Garcia-Valls et al., 2014; Mohamaddiah et al., 2014; Ward and Barker, 2014; Hazarika and Singh, 2015; Alcaraz-Calero and Aguado, 2015; Sugapriya and Jeya, 2015; Alhamazani et al., 2015). However, such monitoring studies have not addressed the area of self-adaptive applications within edge computing frameworks, which represent a new era of cloud computing. This is an important area, since failing to analyze and determine the whole spectrum of monitoring requirements in the right way may lead to a massive software engineering failure for these types of applications.

The primary goal of the present paper has been to explore the fundamental challenges to the evolution of monitoring in the edge computing context that are not well-addressed in academic literature and industries. In order to understand how these challenges can currently be met, we focus on the comparison and analysis of characteristics of existing cloud monitoring technologies to determine their strengths and weaknesses precisely in this domain.

More specifically, the main contribution of this paper can be summarized as follows: (I) providing a systematic analysis of different monitoring concepts within edge computing frameworks; (II) identifying the main open challenges and significant technical issues in such modern monitoring approaches; (III) providing a taxonomy of monitoring requirements needed to support dynamic adaptation of applications within edge computing frameworks; and (IV) presenting the future research directions for monitoring techniques in the adaptation of edge computing applications.

The rest of the paper is organized as follows. Section 2 presents a requirement analysis of monitoring levels within edge computing frameworks. We finish this section with an overview of challenges. Section 3 develops a taxonomy of monitoring requirements for edge computing applications. Section 4 discusses existing widely-used monitoring technologies and the level to which they address the requirements identified from an edge computing viewpoint. The conclusion appears in Section 5.

2. Monitoring levels

To adapt edge computing applications to the changing execution environment and ensure application QoS requirements continue to be satisfied, it is necessary to employ a comprehensive monitoring system able to address the whole spectrum of requirements, pertaining to different levels including (1) the underlying infrastructures (e.g. VM's computing resources, etc.), (2) edge computing platforms (e.g. Docker containers, etc.), (3) network connections between individual application components and (4) application-specific measurements (e.g. service response time, etc.).

This section explains a requirement analysis of all four previously-mentioned monitoring levels for cloud-based applications from an edge computing viewpoint. A modern software engineering discipline provides an approach to design such applications based on a set of different loosely coupled independent components running either in Virtual Machines (VMs) or containers and hence, for completeness, we consider both VM and container levels of virtualization. Emphasis in this section has been put on the importance of monitoring needs for self-adaptive applications from edge computing viewpoint, in order to present a new taxonomy of monitoring requirements for such applications in Section 3.

[4] CoreOS, https://coreos.com/.

[5] Kubernetes, https://kubernetes.io/.

[6] OpenShift Origin, https://www.openshift.org/.

[7] Docker Swarm, https://docs.docker.com/swarm/.

Table 1
Overview of research on different VM-level monitoring systems.

Paper	Goal	Measured metrics (VM-level)
Al-Hazmi et al. (2012)	Monitoring federated clouds	CPU, memory, disk, network usage, etc.
Wood et al. (2008)	Modeling resource utilization of cloud applications	CPU, disk, network usage
Kwon and Noh (2013)	IaaS cloud monitoring	CPU, memory, disk
Meera and Swamynathan (2013)	IaaS cloud monitoring	CPU, memory
Clayman et al. (2010a)	Monitoring federated clouds	CPU, memory, network usage
Caglar and Gokhale (2014)	Intelligent resource provisioning	CPU, memory

2.1. VM-level monitoring

All the physical resources including CPU, memory, disk and network can be virtualized. Multiple VMs can be deployed on a single physical machine and thus share physical resources between each other. Based on the vision of edge computing, it is necessary to have control over a pool of configurable virtualized resources exploited in both the centralized cloud computing layer (the first layer depicted in Fig. 1) and the edge computing layer (the third layer shown in Fig. 1). It should be possible for such resources to be autonomously provisioned and de-provisioned with little, or preferably no intervention of an application provider. In order to have efficient resource utilization and prevent any problems in virtualized resources, monitoring of the VMs used in the cloud datacenters and edge nodes is critical. Performance optimization can be best achieved by efficiently monitoring the utilization of these virtualized resources. Capabilities for monitoring such resources include tools for monitoring mainly usage of CPU, memory, storage and network:

- CPU usage shows the amount of actively used CPU as a percentage of total available CPU in a VM. If the processor utilization reaches 100% and the CPU run queues start filling up, the system has run out of available processing capacity and adaptation action must be taken at that point – or, preferably, before that point, in anticipation.
- Memory usage indicates the percentage of memory that is used on the selected machine.
- Disk usage refers to the amount of data read or written by a VM. Or it can also indicate the percentage of used drive space. Adding additional storage to the VM and allocating it to the appropriate partition can often resolve disk space issues.
- Network usage is the volume of traffic on a specific network interface of a VM, including external and internal data traffic.

Relevant papers that have been published in this area are included in Table 1. We provide more details in the following paragraphs.

Al-Hazmi et al. (2012) adopted a cloud monitoring system to provide Monitoring-as-a-Service (MaaS) usable by both cloud-based application providers and customers which may have different views on the monitoring data in a multi-tenant environment. Their monitoring solution works across federated clouds; however it is restricted to datacenters' monitoring tools. This approach needs all infrastructures to apply the same monitoring system; whereas an important requirement in edge computing frameworks is for a monitoring solution which is not cloud-specific and capable of working across federated testbeds.

Wood et al. (2008) developed a mathematical model to estimate resource overhead for a VM. The proposed model can be adopted for approximating virtualized resource requirements (especially CPU) of any application on a determined platform. Moreover, the model can be used to estimate the aggregated resource needs for VMs co-located on one host. Their solution defines the minimum amount of resources necessary for a VM to avoid performance reduction because of resource starvation. However, their approach is not able to directly measure how application performance (measured, for example, by response time) will vary. Another notable point in this work is that profiling resource usage of virtualized applications is offline and occurs on monthly or seasonal timescales. Therefore, it makes the proposed model less useful due to the dynamic environment of edge computing scenarios.

Kwon and Noh (2013) demonstrated an architecture for a monitoring system consisting of a dashboard to show the real-time resource utilization for servers and VMs. It was mentioned that if CPU, memory, and storage are overloaded, then the virtual servers will not be able to perform their normal function. However, the article did not explain how the experiment could be implemented in practice and therefore it is not completely clear that the proposed solution can be used to improve the performance of VMs, as the authors claim. It should be noted that the proposed monitoring architecture is usable only for a specific kind of virtualization (Xen hypervisors).

Meera and Swamynathan (2013) proposed an agent-based resource monitoring system which provides VM-related information (CPU and memory utilization) to the cloud-based application provider for efficient resource optimization. The proposed monitoring architecture can be improved by adding an alarm feature that triggers if a value breaches a threshold that can be used for many purposes such as failure prediction or Service Level Agreement (SLA) assessment. This architecture is limited by its dependence on centralized coordination. In contrast, with regard to the needs of self-adaptive edge computing solutions, monitoring agents which report the availability of resources should be autonomous.

Clayman et al. (2010a) described a monitoring framework called Lattice which is able to measure and report system parameters of virtual infrastructures for the management of cloud-based services in real-time. This monitoring framework provides necessary libraries along with APIs to implement a customized monitoring system and it could be considered more as a toolkit than a ready-to-use tool. The main functionality of the Lattice monitoring framework is the collection and distribution of measurement data through either UDP or multicast protocol. Therefore, their proposed solution does not include the functionality for visualization.

Caglar and Gokhale (2014) presented an autonomous, intelligent resource management tool called iOverbook usable in heterogeneous and virtualized environments. This tool provides an online overbooking strategy based on a feed-forward neural network model by carefully considering the historic resource usage to forecast the mean hourly CPU and memory usage one step ahead. However, their work could be improved by effective filtering of potential outliers that can quench heavy data transmission overhead especially on edge nodes in large-scale edge computing environments. Also, in order to make the proposed solution capable of working with a high sampling frequency in highly dynamic environments, one step further could be taken to consider various time intervals instead of a fixed hourly rate.

2.2. Container-level monitoring

In comparison with VMs, the use of containers which does not require an Operating System (OS) to boot up as another form of server virtualization (Seo et al., 2014) is rapidly increasing in pop-

Table 2
Common set of container-level parameters.

Container-level metric	Type	Description
rx_bytes	B/s	Bytes received by the container
rx_packets	Pckt/s	Packets received by the container
tx_bytes	B/s	Bytes sent by the container
tx_packets	Pckt/s	Packets sent by the container
cpu_usage	Float	%CPU usage of the container
memory_usage	KB	%memory usage of the container
io_service_bytes_read	B/s	Bytes read from block device by the container
io_service_bytes_write	B/s	Bytes written to block device by the container

ularity. One particular reason for this is that container images have significantly smaller size than full VM images. Along these lines, different container-based virtualization platforms such as Amazon EC2 Container Service (ECS)[8] and Google Container Engine (GKE)[9] have been provided as alternatives to hypervisor-based technologies. Because of the lightweight nature of containers, it is possible to deploy container-based services (e.g. microservices (Kratzke and Quint, 2017)) in different hosting environments faster and more efficiently than using VMs. Moreover, it is easier to pull container images or migrate container instances of application components across computing nodes in the cloud; the more traditional VM-based techniques are too heavyweight to achieve this type of agility that is required in edge computing frameworks. This agility is needed because the migration process is an efficient tool for many purposes such as load-balancing, dealing with hardware failures, scaling systems or reallocating resources. A migration mechanism has to operate automatically to provide almost zero downtime even in the specific situation, for example, when an infrastructure failure occurs in each level of multi-layer edge computing frameworks shown in Fig. 1.

According to the edge computing trend in which cloud environments are becoming more dynamic and workloads vary over time, using this lightweight cloud technology can support self-adaptation of the entire system to address the needs of application providers and users. This support is also due to the ability of particularly interoperable service packaging and orchestration that this technology provides us to bind various software components in order to build the whole application. It means that decentralized edge clouds which move the computation and data management to the edge of the network away from datacenters require a lightweight distribution and orchestration of portable application components such as containerization.

If the system uses containers to run application services in both the centralized cloud computing layer (the first layer depicted in Fig. 1) and the edge computing layer (the third layer shown in Fig. 1), container-level monitoring becomes mandatory.

According to the literature (Stankovski et al., 2016; Preeth et al., 2015; Beserra et al., 2016; Vangeepuram, 2016; Dusia et al., 2015), the common set of container-level metrics to be monitored and useful in the context of application adaptation is shown in Table 2.

Besides this, there are different tools provided just in order to monitor containers and display runtime value of key attributes for a given container, as listed in Table 3.

All container-specific monitoring tools compared in Table 3 provide a REST API to expose statistics about a given container and this remote API can be externally invoked by other entities.

Docker provides a built-in command called *docker stats* which reports runtime metrics and resource usage for a given container. Retrieving a detailed set of metrics is also possible by sending a GET request to the Docker Remote API.[10]

Container Advisor (cAdvisor) is an open-source system that measures, aggregates, processes, and displays monitoring data about running containers. This monitoring information can be used as an understanding of the runtime resource usage and performance characteristics of running containers. cAdvisor shows data for the last 60 s only. However, it supports the ability to easily store the monitoring information in an external database such as InfluxDB that allows long-term storage, retrieval and analysis. InfluxDB is an open-source Time Series Database (TSDB) capable of real-time and historical analysis. Complementing this, Grafana is an open-source Web-based user interface to visualize large-scale monitoring information. It is able to run queries against the database and show the results in an appropriate scheme. On top of cAdvisor, using Grafana and InfluxDB could effectively improve visualizing the monitored parameters collected by cAdvisor in concise charts for any time period.

Prometheus[11] is an open-source monitoring tool as well as a TSDB. It gathers monitoring parameters from pre-defined resources at specified intervals, shows the results, checks out rule expressions, and is capable of triggering alerts if the system starts to experience abnormal behavior. Prometheus uses LevelDB[12] as its local storage implementation for indices and storing the actual sample values and timestamps. Although cAdvisor, in comparison with Prometheus, has been considered as the easier tool for use, it has limits with alerting on occurrence of an identified event when something requires attention. However, both may not properly provide turnkey scalability themselves, capable of handling large number of monitored containers.

Docker Universal Control Plane (DUCP) is a tool to manage, deploy, configure and monitor distributed applications built using Docker containers. This container management solution supports all the Docker developer tools such as Docker Compose to deploy multi-container applications across clusters. High scalability and Web-based interface are some of the key features of DUCP as a Docker native commercial solution.

Scout[13] is another container monitoring tool which has a Web-based graphical management environment, and is able to store at most 30 days of measured metrics. It consists of a logical reasoning engine capable of alerting based on metrics and their associated predefined thresholds. Similar to Scout, there are many commercial solutions to monitor containers with the same characteristics.

Container-level monitoring is currently a hot research topic, as compared to related areas, for example monitoring of cloud infrastructures. Relevant papers that have been published in this area are included in Table 4.

Stankovski et al. (2016) proposed a distributed self-adaptive architecture that applies the edge computing concept with container-based technologies such as Docker and Kubernetes to ensure the QoS for time-critical applications. Their idea is to deploy the containerized application (file upload use case) in different geographic locations in a way that the service is created, served and destroyed for every file upload request. For each container, features of resources required for the host can be allocated upon monitoring data and operational strategies defined by the end-user, application developer and/or administrator.

Preeth et al. (2015) evaluated the performance of Docker containers based on system resource utilization. From their benchmarks, the authors found that container-based virtualization can be compared to an OS running on bare-metal in terms of memory,

[8] Amazon EC2 Container Service, https://aws.amazon.com/ecs/.

[9] Google Container Engine, https://cloud.google.com/container-engine/.

[10] Docker Remote API, https://docs.docker.com/engine/api/v1.21/.

[11] Prometheues, https://prometheus.io/.

[12] LevelDB, https://github.com/syndtr/goleveldb.

[13] Scout, https://scoutapp.com/.

Table 3
Overview of container-specific monitoring tools.

Monitoring tool	License	REST API	Being scalable	Alerting	Time Series Database (TSDB)	GUI
Docker Built-in Tool[a]	Apache 2	Yes	No	No	No	No
cAdvisor[b]	Apache 2	Yes	No	No	No	Yes
cAdvisor + InfluxDB[c] + Grafana[d]	cAdvisor: Apache 2, InfluxDB: MIT, Grafana: Apache 2	Yes	Yes	No	Yes	Yes
Prometheus	Apache 2	Yes	No	Yes	Yes	Yes
DUCP[e]	Commercial	Yes	Yes	Yes	No	Yes
Scout	Commercial	Yes	No	Yes	Yes	Yes

[a] Docker, https://www.docker.com/.
[b] cAdvisor, https://github.com/google/cadvisor.
[c] InfluxDB, https://influxdata.com/time-series-platform/influxdb/.
[d] Grafana, http://grafana.org/.
[e] Docker Universal Control Plane (DUCP), https://docs.docker.com/ucp/.

Table 4
Overview of research on different container-level monitoring systems.

Paper	Goal	Measured metrics (Container-level)
Stankovski et al. (2016)	Adaptation of cloud applications (for time-critical applications)	CPU, memory, disk, network usage
Preeth et al. (2015)	Performance evaluation of virtualization technology	CPU, memory, disk, network usage, etc.
Beserra et al. (2016)	Performance evaluation of virtualization technology (for I/O-bound applications)	Disk
Vangeepuram (2016)	Performance evaluation of virtualization technology (for Cassandra server)	CPU and disk
Dusia et al. (2015)	Support application QoS (for bandwidth-intensive applications)	Network usage

CPU and disk usage. With regard to these three metrics, the performance of Docker approximates the performance of native environment. But, according to the network utilization, the host OS has considerably less network bandwidth compared to the Docker container. However, in this work, one host was allocated to just one container to simplify the experiments. In addition, container performance against other virtualization technologies could be evaluated as a further complement to this work.

Beserra et al. (2016) analyzed the performance of container virtualization in comparison with VM-based virtualization for HPC disk I/O-bound jobs. Based on the evaluation, results showed that containerized environments generally work more effectively than VMs for disk I/O and data intensive applications. Moreover, both virtualization technologies achieved the same performance if there is just one abstraction per physical server.

Vangeepuram (2016) undertook an experimental performance comparison of Apache Cassandra TSDB between Linux container and bare metal. They considered three different workload scenarios: write, read and mixed (read & write) loads. The results of the research work showed that the Cassandra cluster on bare metal consumes less CPU utilization than it does when it is containerized in both write and mixed scenarios, but there is no significant difference for the read scenario. The Cassandra cluster on bare metal has less latency compared to containers in all scenarios. Furthermore, considering the disk throughput, no concrete inference can be drawn from the findings. However, memory utilization was not considered in this work. Monitoring memory consumption would have provided more realistic results, since the performance of Cassandra becomes poor if the system does not have enough memory as it starts to do mostly garbage collection at some point.

Dusia et al. (2015) introduced a mechanism to guarantee network QoS for time-critical applications using Docker. Their implementation is able to prioritize the network access of all containers running on a host in a way that the containers with higher priority will be given more share of the total available network bandwidth. Therefore, in this way, a containerized application that is network bandwidth intensive cannot result in a poor or undesirable execution due to other applications sharing a Docker host. However, authors performed their experiments in a static setting, without considering current dataflow requirements or ongoing buffer status.

2.3. End-to-end link quality monitoring

Cloud-based applications, such as early warning systems, have time-critical requirements, such as minimal delay and jitter tolerance, and require suitable support to achieve guaranteed network-based QoS. This is a challenge because performance is difficult to keep up if the network infrastructure's conditions continuously change.

The idea that some application services are deployed on the nodes at the edge of the network and others on centralized datacenters has raised serious concerns about the network quality of links between these services across an edge computing framework. This is a challenging research area, because it relates not only to live-migration of services between edge nodes and datacenters, but also among different nodes at the same layer in edge computing frameworks.

Network performance for all communications passing through the edge computing framework has to be measured by end-to-end link quality monitoring. Regardless of what network technologies are being used, the edge paradigm contains four types of network connections among application components to be considered:

- Communications between a cloud datacenter and an edge node: new enabling technologies, such as SDN and NFV, provide a basis for applying advanced principles on how networks can be appropriately developed, implemented, deployed and operated between the cloud datacenter and edge nodes in an edge computing framework. In recent years, SDN and NFV open up novel opportunities providing a method for virtualizing network on-demand based on end-to-end connection quality at any time. Network components (e.g., routers, bridges, switches, etc.) can be virtualized by NFV, and thereby it is efficiently possible to dynamically instantiate, migrate and scale up or down network functions such as routing, packet forwarding and firewall services. Alongside NFV, SDN offers a set of APIs and control protocols such as SNMP and OpenFlow that enable the network to be programmable, managed and automated.
- Communications between edge nodes: edge nodes at various geographical locations manage a pool of virtualized resources locally. In this way, collaborative provisioning and content delivery among peered edge nodes helps the application provider improve the entire application performance. The data transmis-

Table 5
Overview of research on different link quality monitoring systems.

Paper	Goal	Measured metrics (Network-level)
Lampe et al. (2013)	Support application QoS (for audio/video streaming services)	Delay
Chen et al. (2014)	Support application QoS (for cloud gaming systems)	Throughput, delay, packet loss
Samimi et al. (2007)	Support application QoS (for communication services)	Delay, packet loss, jitter, etc.
Mohit (2010)	Support application QoS (for communication services)	Throughput, delay, packet loss
Hsu and Lo (2014)	Ensure QoE to the users (for multimedia services)	Throughput, delay, packet loss, jitter
Taherizadeh et al. (2016a)	Ensure QoE to the users (for data streaming services)	Throughput, delay, packet loss, jitter
Cervino et al. (2011)	Support application QoS (for real-time streaming services)	Throughput, delay, packet loss, jitter

sion among these nodes can be performed either by a centralized approach such as SDN, or via a fully distributed method through traditional routing protocols, e.g. OSPF (Verma and Bharadwaj, 2016).

- Communications between/within cloud datacenters: with the remarkable growth in cloud-based applications, the volume of data exchanged between software components in different tiers deployed on cloud datacenters is rapidly increasing. Ensuring that these types of applications are able to offer favorable service quality has been a challenging issue due to runtime variations in network conditions intrinsic to connections between individually replicated and distributed application components across/within different cloud datacenters.
- Communications between IoT object/user and edge nodes: self-adaptive application providers have to dynamically adapt their services to the IoT device and customer's network circumstances to provide high performance and a seamless experience.

According to the literature (Lampe et al., 2013; Chen et al., 2014; Samimi et al., 2007; Mohit, 2010; Hsu and Lo, 2014; Taherizadeh et al., 2016a; Cervino et al., 2011), the most important metrics to be analyzed for network measurement include:

- Network throughput, which is the average rate of successful data transfer through a network connection.
- Network delay, which specifies how long a packet takes to travel across a link from one endpoint or node to another. This metric can also mean Round-Trip Time (RTT) which is the time elapsed from the propagation of a message to a remote place to its arrival back at the source.
- Packet loss, which is when one or more packets of data traveling across a network fail to reach their destination.
- Jitter, which is the variation in the end-to-end delay of sequential received packets. This network parameter is extremely important for real-time applications, e.g. oil exploration or connected vehicle application, as jitter impacts the size of the associated data stream buffers.

As can be seen in Table 5, some efforts have been made to research and build monitoring systems that focus on end-to-end link quality measurement of cloud environments.

Lampe et al. (2013) explained that limitations of the network infrastructure, such as high latency, potentially affect the QoS of cloud-based computer games for a user. The authors conducted their research focusing on network latency measurement and would benefit from additional end-to-end link quality metrics; for example, the effects of network disturbances, such as increased packet loss or fluctuating throughput which are noticeable indicators of network performance within edge computing frameworks.

Chen et al. (2014) performed an extensive traffic analysis of two commercial gaming systems (StreamMyGame and OnLive). The results demonstrate that limitations of bandwidth and packet loss cause a negative effect on the graphic quality and the frame rates in the cloud gaming systems. Alternatively, the network delay does not predominantly impact the graphic quality of the gaming services, due to buffering. The authors focus on the users' perspective in cloud-based gaming systems, and the performance of gaming services could be evaluated from the service providers' perspective as another research area.

Samimi et al. (2007) introduced a model including a network-based monitoring system and enabling dynamic instantiation, configuration and composition of services on overlay networks. The results show that to simplify and accelerate the deployment and prototyping of communication services, distributed cloud infrastructures can be designed and used to dynamically adapt the service quality to the workloads in which the service provider needs a large number of resources. However, when it comes to overlay networks, encapsulation techniques are not without drawbacks, including overhead, complications with load-balancing and interoperability issues with devices like firewalls.

Mohit (2010) mentioned that computation-based infrastructure measurement is not adequate for the optimal implementation of running cloud services. Network-level evaluation of the cloud service is also very important. The author suggests an approach that includes use of different technologies without implementation and detailed information. Moreover, their solution relies on high capacity edge routers which are expensive and consequently cannot be afforded in all use cases.

Hsu and Lo (2014) presented a mapping from QoS to QoE and thereby an adaptation model to translate end-to-end link quality metrics (including delay, packet loss rate, jitter and throughput) into QoE of the end-user in multimedia services running on the cloud. To this end, they proposed a function to evaluate the QoE score after the user watches the streaming video. The results indicate that the network QoS and users' QoE are consistent and linked together. Therefore, service providers are able to apply the proposed autonomous function to calculate users' QoE impression and to rapidly react to the QoE degradation. The proposed approach does not take into account the trade-off between network cost optimization and quality which is a significant factor to be considered in the adaptation of edge computing scenarios (Ahmed and Rehmani, 2017). However, it should be noted that different users have different objectives e.g. conflicting objectives of price and quality.

Taherizadeh et al. (2016a) proposed a non-intrusive network edge monitoring approach which is able to measure critical QoS metrics including delay, packet loss, throughput and jitter in order to adapt service quality experienced by the users for real-time data streaming applications running on edge computing platforms. The authors claim that network edge-specific monitoring knowledge helps application providers accomplish more satisfactory adaptations to the user's conditions (e.g. network status). In this work, the main adaptation possibility includes dynamically re-connecting users to a set of the best reliable servers offering fully-qualified network performance.

Cervino et al. (2011) performed experiments to evaluate the benefits of deploying VMs in clouds for P2P streaming. The authors strategically placed network traffic distribution nodes into cloud infrastructures around the world owned by Amazon. The main goal is to improve the QoS of live-streaming even in P2P video-

Table 6
Overview of research on application-level monitoring systems.

Paper	Goal	Measured metrics (Application-level)
Leitner et al. (2012)	Adaptation of cloud applications (e.g. Twitter)	Response time (time-per-data item), application throughput (data item-per-second), etc.
Evans et al. (2015)	Adaptation of cloud applications (e.g. Twitter)	Total incoming tweets per second, number of channels running in pipeline, etc.
Emeakaroha et al. (2012)	Cloud application SLA violation detection	Response time, application throughput, etc.
Farokhi et al. (2015)	Adaptation of cloud applications (e.g. Web applications)	Response time
Xiong et al. (2013)	Modeling the performance of cloud applications (e.g. Web applications)	Response time and application throughput
Mastelic et al. (2012)	Modeling the performance of cloud applications (e.g. audio and video services)	Response time (render time per frame)
Shao and Wang (2011)	Modeling the performance of cloud applications	Response time, availability of application, etc.
Wamser et al. (2017)	Ensure QoE to the users (e.g. live video streaming services)	Application throughput (frames per second), dropped frames and video quality
Rossi et al. (2015)	Modeling the performance of cloud applications	Response time
Jamshidi et al. (2015)	Adaptation of cloud applications	Response time and application throughput
Rao et al. (2011)	Intelligent resource provisioning	Response time and application throughput
Islam et al. (2012)	Intelligent resource provisioning	Response time

conferencing services by creating network connections among Amazon's cloud datacenters, making use of the low latency and reduced packet loss that exists among its datacenters. The authors assess different network QoS metrics of connections between Amazon datacenters in different regions that provide better network quality than the average Internet connection. Their implementation deals with trans-continental communications and consequently does not address traffic localization including data exchanges between nearby peers just in one area. Since in some cases such as Pokémon GO, (one of 2016s successful games) edge nodes distributed around a country (e.g. Australia) would occasionally need to send data (only a subset of scoring information) up to a central datacenter whereas the game requires constant back-and-forth interactions between the users and the edge nodes in close proximity.

2.4. Application-level monitoring

The edge computing framework itself is application agnostic in that it is not dedicated to a single type of software system or purpose. However, every cloud-based application needs to be extended to include application-level monitoring capabilities to measure metrics that present information about the situation of the service and its performance. Although a large number of research works consider the reliability of the underlying cloud infrastructures, there is still a lack of efficient application-level monitoring techniques to be able to detect and measure QoS degradation. In Table 6, different works to monitor application-level metrics are summarized.

Leitner et al. (2012) proposed the monitoring system called CloudScale which measures the distributed application's performance at runtime and also adopts user-specified scaling policies for provisioning and de-provisioning of virtual resources. Their proposed event-based approach models the workload behavior, supports multi-dimensional analysis, and defines the adaptation action. However, it considers only elasticity which often could be to increase and decrease the total number of computing nodes in the resource pool, regardless of application topology or reconfiguration.

Evans et al. (2015) used container-based virtualization to run a Twitter analysis application called Sentinel. The application consists of multiple containerized components distributed in the cloud and can provide Docker container reconfiguration on demand as well as real-time service monitoring to inform the reconfiguration module to restructure the application based on changing circumstances (load, etc.). The proposed system is capable of being scalable for running components to be dynamically duplicated in order to share the workload.

Emeakaroha et al. (2012) implemented the monitoring framework called CASViD which is general purpose and supports the measurement of low-level system metrics for instance CPU and memory utilization as well as high-level application metrics which depend on the application type and performance. The results imply that CASViD, which is based upon a non-intrusive design, can define the effective measurement interval to monitor different metrics. It can offer effective intervals to measure different metrics for varied workloads. As the next step, it is possible to improve this framework in order to support multi-tier applications. This is challenging, due to the distributed nature of edge computing applications nowadays, each application has different types of components with different application-level metrics.

Farokhi et al. (2015) proposed a fuzzy autonomic resource controller to meet service response time constraints by vertical scaling for both memory and CPU without either resource over- and under-provisioning. The controller module autonomously adjusts the optimal amount of memory and CPU needed to address the performance objective of interactive services, such as a Web server. Since the maximum memory or CPU capacity is limited, the proposed system could be extended to consider both vertical and horizontal scaling to be able to afford unlimited amount of workload possibly generated in large-scale edge computing scenarios.

Xiong et al. (2013) proposed a model-driven framework called vPerfGuard to achieve automated adaptive application performance. This approach, which is capable of identifying performance bottlenecks, has three modules: (1) a sensor element to collect runtime system metrics as well as application-level metrics e.g. response time and application throughput; (2) a model building element that enables the creation of a customized performance model showing the correlation between system metrics and application performance; (3) a model updating element to automatically detect when the performance model needs to be changed. A potential downside to this approach is that if the performance analyst does not have enough knowledge or experience to define a proper performance metric to make the model, their mechanism may not be useable. Moreover, outliers can have huge effects on the regression, limiting the dependability of the metrics obtained as a basis for adaptation in some contexts.

Mastelic et al. (2012) discussed how the CPU or memory usage affects the response time, which in their case, is the render time per frame. Their monitoring system includes the consumption of all processes belonging to the application. These processes have the same parent process and hence by listing a list of process IDs (PIDs) for the monitored application, it is possible to sum up the resource consumption of all processes belonging to the application and calculate total resource consumption at a certain point in time. This model could benefit from being extended to include other types of metrics; for instance network-level parameters, since monitoring and management of various real-time data streaming applications including audio and video streaming services is a big data challenge.

Shao and Wang (2011) proposed a performance guarantee approach based on a Runtime Model for Cloud Monitoring (RMCM). In this work, a performance model is constructed upon runtime monitored data using the linear regression algorithm. These relevant metrics include the resource allocated to the VM where the application resides, the number of co-existing applications on the same VM, the actual resource occupation by the application, the workload and so on. The results show that the performance model can be effective for controlling the provisioning approach to attain specified performance objectives. Because of the manual installation and configuration of monitoring agents in this work, non-functional requirements of monitoring, such as scalability and migration, may not be satisfiable. A further direction to pursue could be also to consider how to find the effective measurement intervals.

Wamser et al. (2017) focused on the live-migration of service instances for HTTP-based video streaming to cope with the impact of user mobility. They used an edge computing environment to obtain fast replacement of cloud services across different edge nodes if a user perceives poor video quality. In this work, supported by the INPUT project,[14] a Deep Packet Inspection (DPI) monitoring tool was used to measure three application-level metrics including frames per second, dropped frames and video quality, since these metrics have a positive correlation with the user's QoE. When any of predefined thresholds for these metrics is violated for a specific period of time, the service has to be migrated from the current edge node to another one.

Rossi et al. (2015) introduced a model to estimate the response time of cloud applications according to the Linux OS's counters, for example LoadAVG. One of the most important reasons to estimate application-level metrics upon low-level metrics (e.g. CPU load) is that monitoring application-level metrics (such as response time) could cause overhead in both network channel and computing resources. Moreover, monitoring high-level metrics can have privacy implications for users. The results show that the load values given by LoadAVG accompany the application response time behavior, which leads to a strong positive correlation between the two behaviors. Their work only considers LoadAVG, and hence this model could be developed more completely by exploring other counters such as *iostat* and *netstat*.

Jamshidi et al. (2015) presented a self-learning adaptation technique called FQL4KE which is a fuzzy control method based on the Reinforcement Learning (RL) algorithm for learning optimal elasticity policies. This approach aims at automating the scaling process without leveraging any *a priori* knowledge on the running cloud application. The proposed architecture includes a learning module which constantly upgrades the knowledge base of the controller by learning adaptation rules suitable for the system. However, the proposed approach cannot deal with time-varying goals. If system goals change, the controller has to relearn everything from the beginning. A more fundamental problem in real world environments is the fact that the number of situations can be enormous and therefore the learning procedure could become impractical, due to time constraints in new computing paradigms.

Rao et al. (2011) used a distributed RL mechanism called iBalloon for self-adaptive VM resource provisioning in which monitoring is essential for autonomic orchestration and adaptation. Nowadays, cloud infrastructures offer elastic resources by horizontal or vertical scaling solutions to adapt the application performance to the changing workload. However, such current scaling approaches which utilize only infrastructure-related monitoring data may cause severe performance drops during workload variations at runtime. The authors claim that monitoring only infrastructure-level metrics such as memory and bandwidth without taking into account how application performance is behaving (application-level monitoring) at runtime would complicate the resource provisioning problem due to the lack of detailed measurement. In their work, according to the proposed vertical scaling approach, each VM is able to adjust its resource allocation in terms of CPU, memory, and bandwidth. The iBalloon architecture includes three fundamental elements: (1) a host-agent which is in charge of allocating resources to the VMs; (2) an app-agent which includes the monitoring part of iBalloon and reports runtime information about application performance; and (3) a decision-maker which hosts an RL agent placed at each VM to perform automatic resource capacity adjustment. However, iBalloon is limited because it does not consider other virtualized resources e.g. storage, nor does it support other adaptation actions e.g. migration to improve application performance. This is an issue because in order to manage an application deployed on an edge computing framework, it is necessary to consider the migration of application components among heterogeneous infrastructures (Desertot et al., 2005).

Islam et al. (2012) developed a proactive cloud resource management approach in which linear regression and neural networks have been applied to predict and satisfy future resource demands. The research problem in this work actually is to analyze time series monitoring data to extract a prediction model and other characteristics of the monitoring data. The proposed performance prediction model estimates upcoming resource utilization (e.g. aggregated percentage of CPU usage of all running VM instances) at runtime and is capable of launching additional VMs to maximize application performance. The authors considered predictive accuracy based on the application performance in terms of response time. This approach provides distributed scaling and can be enhanced to address the resource allocation of a single VM also. At present, only CPU utilization is used to train prediction model and their approach could further include other types of resources, e.g., memory, disk and bandwidth.

2.5. Challenges to the evolution of monitoring in the edge computing context

Based on our analysis, the following are the most important current challenges in monitoring adaptive applications within edge computing frameworks:

- Mobility management: When a client device is moving, due to varying network parameters of link between end-user and edge node such as jitter, delay, bandwidth and so on, the application QoS can increase and decrease rapidly, in a way that is potentially difficult to predict (Ahmed and Ahmed, 2016). As the location of users or end-devices may change over time, dynamic service migration has gained increasing attention in the context of the edge computing paradigm to deliver always-on services.
- Scalability and resource availability at the edge of the network: Lightweight jobs are most likely processed at the edge of the network, and hence edge nodes may have hardware capacity limits. Simultaneously, it should be guaranteed that these

[14] The INPUT project, http://www.input-project.eu/.

nodes are able to accommodate the increasing demand for delivering services and growing network traffic volume. Edge nodes should ensure the availability of the service regardless of the number of end users' client devices at the edge network (Ahmed and Ahmed, 2016).

- Prior knowledge: Supporting the QoS constraints requires the pre-knowledge of execution environment such as underlying infrastructure and the configuration of application components on the network (Xiao et al., 2016). An adaptation technique is fully advantageous, if it does not require relying on the knowledge provided by previous experiences.
- Data management: In large-scale environments, monitoring probes are generating massive amounts of collected data to be aggregated, processed and stored. Consequently, it poses other challenges for instance the utilization of distributed datacenters with more bandwidth (Zhao et al., 2015) and flexible integration capabilities (Esposito et al., 2015).
- Coordinated decentralization: The challenge in building a decentralized system is to ensure that all different application components collectively move the whole system towards a common goal.
- Saving expense, time and energy: On-demand resource allocation should be flexible enough in order to dynamically assign infrastructure according to the needs of modern self-adaptive cloud applications, and an important attribute of such is to reduce their cost. However, cost optimization should not result in QoS/QoE degradation. What should be noted here is that application providers need to take into account where the adaptation logic is inserted: separately, or together with the monitoring components. For example, whether monitoring modules of the application are in charge of considering network cost optimization or it is a task of self-adaptation engine.
- Interoperability and avoiding vendor lock-in: The vendor lock-in situation is generally considered as a disadvantage in cloud computing (Toosi et al., 2014). Recently, significant attempts have been made to address this challenge by driving standardization of edge computing. For example, the open-source EdgeX Foundry project[15] which has started in 2017 supported by the Linux Foundation is aimed at developing a vendor-neutral framework for IoT edge computing. Similarly, the OpenFog consortium[16] has been founded by tech-giants such as Dell, Cisco, Intel, Microsoft and ARM in collaboration with Princeton University to drive standardization of fog computing. This consortium is aimed at leveraging cloud, edge and fog architectures to enable IoT scenarios.
- Optimal resource scheduling among edge nodes: A scheduling mechanism must be intelligent enough to guarantee responses upon the uncertainty of the runtime environment (Lee and Lee, 2015) such as changing workloads, users' channel diversity, their unstable network conditions, user mobility and so on.
- Fault tolerance: The application should continue to operate under the presence of a fault (Chang et al., 2014), such as losing control over edge nodes or undetermined latency. This requires elements of decentralized control or off-line detection and recovery. Fault tolerance has received substantial attention for real-time systems due to their safety critical nature.
- Proactive computing: The new paradigm is autonomously triggering decisions and actions by anticipating future states. To achieve this vision, time constraints must be taken into account and it usually involves dealing with large amounts of historical and streaming data (Fournier et al., 2015).
- Replication of services: In the context of edge computing, services can be replicated running on multiple geographically distributed cloud infrastructures (Farris et al., 2017). Replication of servers has its own technical issues such as temporary inconsistencies to be considered. Furthermore, different companies have different regulations of using (or not using) new technologies e.g. internal legislation on the location of data storage.
- Container security: Using container-based virtualization undoubtedly supports the edge computing paradigm, but poses new security threats. For example, containers are able to communicate directly with the host kernel that can lead to security vulnerability in the system.
- Non-specific edge nodes: In the multi-layer edge computing framework shown in Fig. 1, edge nodes are meant to be a set of heterogeneous computing platforms possibly not as powerful as cloud datacenters. However, the current industry rarely comes up with a solution for providing general-purpose edge nodes involved in computation and data analytics. In this regard, the LightKone project[17] has started in 2017 to move computation out of datacenters and directly on the edge of the network. As a further example, the main vision of another project called Open Edge Computing (OEC)[18] is that all nearby components such as WiFi access points, DSL-boxes, base stations would be capable of offering resources through standardized, open mechanisms to any types of applications to perform computation at the edge. In practice, the edge nodes typically are not convenient for handling the workload as general-purpose computation. Therefore, similar to these nodes, their monitoring mechanisms are generally subject to the proprietary use of a specific technology, and hence they are not able to address multi-purpose needs.

2.6. Summary

In order to simplify the decision-making self-adaptive mechanism in the edge computing environment, monitoring solutions collect information from different levels. To this end, in addition to monitoring virtualized resources (e.g. CPU, memory, disk, etc.), it is important to consider other levels of monitoring, including container, end-to-end network quality and application. In the current section, the papers have been chosen to be reviewed in a way that all the important functional and non-functional monitoring requirements for adaptive applications within edge computing frameworks would be mentioned. This section also indicated recent challenges and future research directions needing to be explored in this field. From the analysis of the literature in the current section, we derive a new taxonomy of requirements for the development and deployment of relevant monitoring technologies within edge computing frameworks, which we present in Section 3.

3. Taxonomy of monitoring requirements in edge computing scenarios

Drawing on the discussion in Section 2 of cited literature, Table 7 presents a taxonomy of monitoring requirements needed to support dynamic adaptation of applications orchestrated upon edge computing frameworks. In this table, there are also 10 common functional requirements which are essential for all types of monitoring systems within edge computing scenarios. Based on this taxonomy, in Section 4 we analyze monitoring tools, to define their challenges and strengths.

[15] The EdgeX Foundry project, https://www.edgexfoundry.org/.

[16] The OpenFog consortium, https://www.openfogconsortium.org/.

[17] The LightKone project, https://www.lightkone.eu/.

[18] The Open Edge Computing project, http://openedgecomputing.org/.

Table 7
Taxonomy of requirements for monitoring systems within edge computing frameworks.

Taxonomy of monitoring requirements within edge computing frameworks		
Functional requirements	Common in all monitoring levels	• Usable by the provider and customers in a multi-tenant cloud environment • Provide the functionality for visualization • Able to filter measured values to diminish data exchanges • Able to be tuned to any desired monitoring time interval • Include capability of long-term storing of measured values • Support scaling adaptation policies for large-scale dynamic environments • Able to set up automated alerts • Support different adaptation actions e.g. service migration, etc. • Automatic installation and configuration of monitoring system • Able to be customized based on monitoring needs
	Both VM level and container level	• Independent from underlying cloud infrastructure provider • Quickly react to the dynamic resource management changes over time • Support monitoring of all types of hardware virtualizations • Offer an API to expose monitoring data
	End-to-end link quality level	• Investigate the whole range of end-to-end network QoS properties • Support on-demand network configuration • Able to reach the device in spite of filters and firewalls • Consider user's link conditions (e.g. network quality)
	Application level	• Able to deal with the application topology and reconfiguration • Define effective measurement interval upon the application performance • Support multi-tier applications • Able to be adapted with time-varying application adaptation goals
Non-functional requirements	• Scalability • Non-intrusiveness • Interoperability • Robustness • Live-migration support	

The functional requirements which are needed for basic monitoring within edge computing frameworks and hence are common in all monitoring levels are summarized in Table 7 and described below:

- **Usable by the provider and customers in a multi-tenant cloud environment:** This requirement means to have the ability of defining multiple roles and views for various types of users with different permissions to access monitoring data. Especially in a multi-tenant provisioning platform (He et al., 2012) where multiple tenants potentially with various QoS values are sharing the same infrastructures and application instances, different tenants should be able to measure parameters and gain access only to the information that pertains to them.
- **Provide the functionality for visualization:** Visualization is a key functionality for analysis of events in dynamically changing environments, such as edge computing scenarios. It provides a powerful interface between the monitoring data stored in the TSDB and the human brain.
- **Able to filter measured values to diminish data exchanges:** It seems important to provide threshold-based filtering capabilities on the monitored nodes to reduce the runtime communication overhead for the monitoring data transmission and storage.
- **Able to be tuned to any desired monitoring time interval:** Any custom time interval can be set within the specified number of seconds, minutes, hours, days or even weeks. The length of monitoring interval is required to ensure reliability, to avoid overhead, and to prevent losing control over the running environment during adaptation actions.
- **Include capability of long-term storing of measured values:** This functionality is one of core competencies to building a monitoring approach that is optimized for the storage and retrieval of monitoring data, so that (for example) the data can be used to inform future adaptation strategies.
- **Support scaling adaptation policies for large-scale dynamic environments:** Elasticity management of services within edge computing frameworks that is able to support scaling policies even in a large-scale environment needs a scalable monitoring solution, which still is an open issue that is left largely unsolved by many of the present monitoring systems. Different characteristics of a monitoring solution such as data storage mechanism, communication protocol for data collection, the resource consumed to perform the monitoring activity and ability of automatic self-configuration to tune the monitoring system over time may affect the scalability of applications in large-scale dynamic environments.
- **Able to set up automated alerts:** In edge computing scenarios, there is often a need to be able to create custom alert rules that meet particular criteria. For example, a monitoring solution should be able to trigger alerts if a given VM or container instance starts to behave irregularly, or if a metric reaches its associated threshold.
- **Support different adaptation actions e.g. service migration, etc.:** Within edge computing frameworks, monitoring systems should deal with uncertainties imposed by application recontextualization (e.g. dynamic IP address management) due to adaptation actions such as VM or container live-migration.
- **Automatic installation and configuration of monitoring system:** Monitoring systems have to be able to automatically detect when a new VM, container or application instance is created due to scaling up elasticity actions. Auto-discovery can be considered as an approach to automatic installation that refers to the process of detecting new devices in a cloud environ-

ment and then performing ongoing monitoring on these devices without human intervention. Similarly a running VM, container or application instance might cease to exist due to scaling down elasticity. Automatic installation and configuration of any monitoring system is a necessity to support scaling adaptation policies in large-scale edge computing environment.
- **Able to be customized based on monitoring needs:** It is necessary to support the customizability of monitoring solutions such as metric extension (incorporate and start measuring any new particular metric) which allows covering conditions particular to a specific environment, therefore providing monitoring approach with a comprehensive view of the execution environment.

Functional requirements at both VM and container levels for monitoring systems within edge computing frameworks are as follows:

- **Independent from underlying cloud infrastructure provider:** The management of federated cloud environments in edge computing scenarios needs interoperable monitoring to share information among heterogeneous frameworks. While it is easy to design a cloud-specific monitoring platform, implementing a generic monitoring solution able to work with multiple cloud infrastructure providers remains a challenging issue.
- **Quickly react to the dynamic resource management changes over time:** This functional requirement is a process, especially for time-critical edge computing applications, through which the monitoring solution rapidly involves detecting and collecting information about the changing environment. Compared to the centralized cloud datacenter approach, edge computing needs a more agile monitoring system especially because the edge of the network is a highly dynamic environment where end-devices may frequently become available/unavailable, devices are moving, or edge nodes change states over time.
- **Support monitoring of all types of hardware virtualizations:** Cloud monitoring solutions within edge computing frameworks should cover all kinds of hardware virtualizations as well as operating systems across federated clouds.
- **Offer an API to expose monitoring data:** All monitoring tools working at VM, container, network link or application must be able to provide an API to expose runtime statistics about a monitored entity and this remote API should be externally accessible by other entities.

The functional requirements at network link quality level for monitoring systems within edge computing frameworks are explained as follows:

- **Monitor the whole range of end-to-end network QoS properties:** With regard to end-to-end link quality-level measurement, QoS attributes change constantly and so network-layer parameters (mainly network throughput, delay, packet loss and jitter) need to be closely monitored in the edge computing environment.
- **Support on-demand network configuration:** Monitoring solutions within edge computing frameworks must be able to support programmable networks which help in managing and controlling virtual network resources dynamically.
- **Able to reach the device in spite of filters and firewalls:** It is usual that specific types of traffic such as ICMP or SNMP packets are filtered in private administrative domains due to various security concerns. In such cases, the monitoring solution should automatically change its mode of operation via different communication protocols to be able to find the user's location, reach the device and then measure end-to-end link quality metrics to the user. To this end, the first requirement is that the network monitoring solution should be able to access the IP address (taking into account any network address translation) of the user's device.
- **Consider user's link conditions (e.g. network quality):** Within edge computing frameworks, beneficial dynamic adaptations to the user's network conditions can be accomplished by utilizing network edge-specific monitoring information. Monitoring and identifying the network quality of connections between end-users and application servers makes decision or control tasks possible which can continuously adapt the deployed service for optimal performance. For instance, in situations where there exists more than one server to provide the service at the edge of the network, the monitoring solution can contribute towards choosing the best application instance to connect to clients based on their network edge conditions such as higher resolution via more stable connection for streaming applications.

Other functional requirements particularly at application level for cloud monitoring systems in edge computing environments have been listed below:

- **Able to deal with the application topology and reconfiguration:** It would be better for the monitoring system to be aware of the topology and any reconfiguration of cloud-based applications due to adaptation actions during the execution within edge computing frameworks in order to support management of metrics collection and various analytics. For example, a monitoring system may know where distributed services together as the whole application are running and how they are connected to each other.
- **Define effective measurement interval upon the application performance:** Measurement intervals for monitoring different types of applications according to their runtime performance should be optimally determined. Short measurement intervals may negatively affect the intrusiveness of monitoring tools, while slow sampling rates might diminish the accuracy of monitoring information.
- **Support multi-tier applications:** Within edge computing frameworks, in contrast to traditional centralized cloud architectures, a single request sent by an end-user leads to multiple interactions among distinct application tiers across different locations. Distinguishing a multi-tier application's response time as individual constituent durations is critical for application performance diagnosis in edge computing environments. Therefore, any monitoring solution in edge computing frameworks should be able to collect monitoring parameters from multiple tiers of the application. Moreover, the future monitoring generation may also consider multi-tier aspects of the application (e.g. monitoring information about different dynamic provisioning policies at each tier) and the interactions between different tiers.
- **Able to be adapted with time-varying application adaptation goals:** If goals of application adaptation change over time, the monitoring solution may need to be dynamically adapted, while keeping the application running during the monitoring upgrade. In these cases, the implementation of a monitoring approach that collects an individual application-specific metric needs to be dynamically reprogrammed, for example, because of a new different definition of application response time.

From the edge computing point of view, all the important non-functional requirements of a monitoring system required to support dynamic adaptation of edge computing applications, as discussed in Section 2 on reviewed literature, are listed below:

- **Scalability:** A scalable monitoring system can handle a remarkable number of monitored resources and services

(Clayman et al., 2010b). This monitoring feature is a very significant property in edge computing environments because of the necessity of managing a wide variety of parameters that need to be monitored across different layers of framework. The runtime orchestration of VMs or containers in edge architectures possibly including thousands of nodes requires a monitoring solution to be scalable to deliver the monitoring data in a flexible and timely manner. Due to the distributed nature of edge computing applications, existing centralized monitoring tools that lack scalability are not suitable because they fail to distribute the monitoring load, which leads to a single point of failure (Xu et al., 2016).

- **Non-intrusiveness:** The edge computing viewpoint attempts to provide a set of application services in a lightweight, simple way and any edge computing monitoring solution should follow this lightweight methodology. Therefore, a monitoring implementation should take a non-intrusive approach because of the necessity of being lightweight to the ordinary flows of application and infrastructure (Taherizadeh et al., 2016b). Consequently, a low overhead monitoring tool, achieved by adopting minimal processing, memory capacity and communication traffic, is an essential entity for increasing efficiency in such environments (Agelastos et al., 2014).
- **Interoperability:** Edge computing aims at the automatic and cooperative deployment and composition of application services, interconnected over both centralized cloud and edge infrastructures within highly distributed or potentially federated environments. Companies can replicate their application components in facilities hosted by different cloud infrastructure providers to balance services or increase availability and reliability or decrease response time under various network conditions and varied amounts of traffic (Grozev and Buyya, 2013). Therefore, what will be needed in the future is interoperability of monitoring systems for highly adaptive cloud applications. Unfortunately, monitoring tools prepared by IaaS providers usually are specific to the underlying infrastructure and are not able to monitor an application running on other cloud providers' infrastructures (Alhamazani et al., 2015).
- **Robustness:** Individual parts of a monitoring system themselves, or even network links connected them together, may fail in ways that probably prevent the monitoring system from keeping up and running continuously. One of the significant challenges for modern monitoring tools is that they should be capable of detecting vulnerabilities in an environment and be able to adapt to a new situation in order to continue its operation (Fatema et al., 2014). Based on this monitoring aspect, it follows that considerable attention is needed to develop more robust cloud monitoring tools which help to cope with different failure scenarios during execution. Especially when it comes to the edge computing paradigm, each application component can have a conversation path to other components, which makes the overall system an intricate network of communications. Therefore, analyzing and solving problems for a distributed, federated system can potentially be cumbersome and infeasible without robust monitoring tools.
- **Live-migration support:** Nowadays, virtualization technologies offer a variety of resource management options such as VM/container creation, deletion, and live-migration (Liaqat et al., 2016). With the growing maturity of edge computing frameworks, the demand for highly available applications, consisting of a set of independently deployable, modular, small services, is increasing. To this end, live migration of services is a highly desirable feature of such solutions. In a live-migration scenario, virtualized services can migrate from a host to another one at any time without stopping operations. Accordingly, the challenge in this regard is how the monitoring system can be able to adjust to the new environment and changing network conditions required to prevent faults (Toosi et al., 2014).

4. Analysis of monitoring tools based on identified taxonomy

As explained before, there exist four levels of monitoring (VM, container, end-to-end link quality and application) needed for self-adaptive edge computing applications. In this regard, the current section has been divided into two subsections in order (I) to describe different widely-used cloud monitoring tools to contribute to the taxonomical analysis and (II) to compare their features based on the taxonomy presented in Section 3 respectively. These comprehensive presentations offer the opportunity to achieve appropriate technical conception of monitoring tools operational within edge computing frameworks. Additionally, the information discussed helps to choose the most appropriate monitoring tool for developing a fully qualified application using edge computing frameworks based on different adaptation objectives.

4.1. Cloud monitoring tools

There are many tools that offer continuous monitoring and visibility for cloud applications. The next subsections describe different types of widely-used monitoring tools usable in edge computing frameworks and outline advantages and disadvantages of each.

4.1.1. Zenoss[19]

Zenoss is an open-source agent-less monitoring platform based on the SNMP protocol and it monitors networks, servers, applications and services. The main functionality of Zenoss is monitoring principal characteristics of cloud such as availability, inventory, configuration, performance and events which are related to the system. It has an open architecture to enable consumers to customize it based on their needs (Gupta, 2015). This collection/monitoring tool is widely-used in enterprise solutions and provides a user interface by which users can configure and monitor the system (Telesca et al., 2014). It also provides statistics about the number and identifier of hosts and tenants available in the system.

Zenoss is able to use predictive thresholds to protect the edge of the network. In this way, it is able to recognize if an edge node's network interface or link suddenly has considerably large traffic. Furthermore, it is able to filter predefined packets (e.g. based on network geography) away from the nodes at the edge of the network. However, it is not robust enough and cannot support the live-migration of services that are essential within edge computing frameworks. Moreover, the product has a limited open-source version for monitoring and the full version requires payment, restricting its applicability in research contexts.

4.1.2. Ganglia[20]

Ganglia (Massie et al., 2004) is a robust distributed monitoring tool capable of being scalable for high-performance computing environments e.g. grids and clusters. It is now being extended to private and public cloud monitoring (e.g. via sFlow[21]). Also, there are cloud-based tools which are integrated with Ganglia, such as Apache Cassandra database usable in edge computing use cases (Confais et al., 2016). However, this monitoring platform does not seem particularly to be focused on edge computing so far. This monitoring system enables the users to have a look at runtime and historical measured data (such as memory utilization and CPU

[19] Zenoss, http://www.zenoss.org/.

[20] Ganglia, http://ganglia.info/.

[21] sFlow, http://blog.sflow.com/2010/10/ganglia.html.

load) for all running VMs/machines that are being monitored. Ganglia uses widely used technologies and protocols such as XDR (External Data Representation), XML (Extensible Markup Language) and RRDtool to store and visualize time series monitoring data. Its implementation has been designed to run on different operating systems and processor architectures, and it has been currently used on the large number of clusters all over the world. Since Ganglia is mainly designed to collect infrastructure monitoring data about machines in a high-performance computing cluster and display them as a series of graphs in a Web interface, it has a drawback for edge computing as it is not appropriate for bulk data transfer (no congestion avoidance, windowed flow control, and so forth).

4.1.3. *Zabbix*[22]

The Zabbix monitoring solution (Tader, 2010) is designed for a server/agent architecture. The Zabbix server runs on a standalone machine to be able to collect and aggregate monitoring data sent by the Zabbix agents. BonFIRE[23] is one the main projects which this open-source monitoring software implementation is designed for. The Zabbix solution supports an alerting system that triggers if predefined events and conditions happen, such as if the memory utilization is over 80%. These alarms are beneficial as the triggers initiate adaptation plans, such as elasticity actions. SQL databases are used to store measured metrics, and a Web front-end and an API are provided to access data. The Zabbix monitoring tool is primarily implemented to monitor network parameters and network services. The Zabbix agent is quite resource efficient and it can reside on edge nodes in a way that is non-intrusive to edge network functions. Because the native Zabbix agent has been developed in the C language it has a relatively small footprint. However, robustness of Zabbix could be inefficient. It could become unstable and requires restarting in some occasions (Simmonds and Harrington, 2009). As another important disadvantage to be considered, auto-discovery characteristic of Zabbix can be inefficient (Murphy, 2008). For example, sometimes it may take more than five minutes for Zabbix to discover that a host is no longer available in the network. This limitation in time can be a serious problem for any runtime self-adaptation scenario, especially if conditions are changing rapidly.

4.1.4. *Nagios*[24]

The Nagios monitoring system is an open-source solution to monitor network and infrastructure resources. It provides a notification system for resources such as machines, switches, networks and so on. Nagios is able to alert administration if anything fails and it also notifies the system if an issue has been detected or settled. For Nagios to be deployed, it requires complex manual configuration (Mongkolluksamee, 2010; Issariyapat et al., 2012) and also scalability could not be seen as its strong point. Nagios uses many buffers, pipes and queues that could cause bottlenecks if the system were to monitor a large-scale cloud environment. In this context therefore, considerable modification would be necessary. As a consequence, it may not be suitable, as it stands, as an appropriate monitoring tool in dynamic environments such as edge computing frameworks.

4.1.5. *OpenNebula*[25]

OpenNebula is a complete cloud management platform. The monitoring part monitors the cloud infrastructure to measure the state of resources, such as VMs and physical machines (Gorbil et al., 2014). OpenNebula's monitoring tool collects individual metrics via several static sensing elements, called monitoring probes, which are running on the resources. The collection mode can use a push or pull method. In push mode, the initiator is the monitoring probe. The monitoring probe can easily send QoS information only when it detects that the changing metrics are greater than their thresholds. In pull mode, the monitoring manager repeatedly queries each probe. This approach is appropriate for keeping maximum consistency between the probes and the monitoring manager if we are dealing with a real-time environment involving a small number of nodes but it may not prove scalable for high update frequencies or large-scale infrastructures. OpenNebula enables users to customize and create simple monitoring probes which apply just the pull mechanism to send the measured information. OpenNebula offers the OneGate component to allow VMs to push monitoring data to the manager in order to collect application-level parameters, recognize the problems in running applications. Note, however, that this module has been designed for only small-scale cloud environments.

OpenNebula is able to support VM migration and can be generally considered as a solution for dynamic environments such as VM-based use cases posed at the edge of network (Minerva and Crespi, 2017). The OnLife Project (Montero et al., 2017) using OpenNebula proposes an edge computing design to dynamically migrate computing services closer to the users in order to ensure low latency interactions between IoT services and users. However, the preliminary barrier to adoption of OpenNebula monitoring part is cloud interoperability as the main field that needs to be addressed.

4.1.6. *PCMONS*

Private Cloud MONitoring System (PCMONS) (Chaves et al., 2011) is a monitoring system aimed at meeting the need of open-source monitoring technologies for private clouds. The monitoring approach is compatible with the Eucalyptus IaaS platform. The main feature of PCMONS is the ability to be extensible in order to adapt to a new environment, such as incorporating new monitoring metrics. However, it has several disadvantages; as PCMONS is a Nagios module, it inherits Nagios performance and scalability issues that preclude applicability to huge cloud infrastructures; and it works only for monitoring infrastructure in private clouds.

4.1.7. *DARGOS*

DARGOS (Povedano-Molina et al., 2013) is a decentralized resource monitoring solution specifically designed for cloud computing infrastructures. DARGOS uses Node Monitor Agents (NMA) and Node Supervisor Agents (NSA). The NMAs are responsible for gathering monitoring information from the VMs and forwarding it to the NSA. The NSAs collect measured information received from monitored resources and they are able to send monitoring data to the cloud administration. DARGOS should be running on the OpenStack platform now. This is the reason why the OpenStack platform has been changed to support DARGOS (Nova project). Consequently, DARGOS is mainly confined to the cloud infrastructure it has been integrated with as it depends on the architecture of the OpenStack Nova implementation. Moreover, DARGOS is neither robust nor does it support live-migration of services that are neglected for edge computing frameworks. The set of metrics monitored by DARGOS is currently quite limited due to its early development status.

4.1.8. *Lattice*

The RESERVOIR[26] project (Clayman et al., 2012) provides technologies by which services and resources can be efficiently provisioned, monitored, managed and migrated across federated clouds

[22] Zabbix, http://www.zabbix.com/.

[23] BonFIRE project, http://www.bonfire-project.eu/.

[24] Nagios, https://www.nagios.org/.

[25] OpenNebula, http://www.opennebula.org/.

[26] The RESERVOIR project, http://cordis.europa.eu/project/rcn/85304_en.html.

Table 8
High-level analysis of functional requirements for cloud monitoring tools.

Tool	Open source	License	Collection	VM monitoring	Container monitoring	End-to-end link quality monitoring	Application monitoring	Data storage method	GUI
Zenoss	Yes	GPL	Pull	Yes	Yes	Yes[a]	Yes	ZODB, MariaDB, HBase, MySQL	Yes
Ganglia	Yes	BSD	Push/Pull	Yes	No	No	Yes[b]	RRDtool	Yes
Zabbix	Yes	GPL	Push/Pull	Yes	Yes	Yes	Yes	Oracle, MySQL, PostgreSQL, SQLite	Yes
Nagios	Yes	GPL	Pull	Yes	No	Yes	No	Flat file, MySQL	Yes
OpenNebula	Yes	Apache 2	Push/Pull	Yes	No	No	No	SQLite, MySQL, Apache Cassandra	Yes
PCMONS	Yes	New BSD	Pull	Yes	No	No	No	Flat file, MySQL	Yes
DARGOS	Yes	Apache 2	Push/Pull	Yes	No	No	Yes[c]	Nova and Neutron DBs	Yes
Lattice	Yes	Apache 2	Push	Yes	Yes	No	Yes	Distributed Hash Table (DHT)	No
JCatascopia	Yes	Apache 2	Push/Pull	Yes	No	No	Yes	Apache Cassandra, MySQL	Yes
Tower 4Clouds	Yes	Apache 2	Push	Yes	No	No	Yes	InfluxDB, Graphite	Yes

[a] Zenoss has developed the Fping ZenPack as an extension to monitor network connections.

[b] However Ganglia is able to monitor built-in metrics (e.g. load average, CPU utilization, disk free, etc.), it is possible to extend Ganglia's metric library to measure application-specific metrics.

[c] DARGOS implements built-in application monitoring solutions to measure the status of Apache web server and MySQL database server. The application-related statistics to be monitored are predefined such as the number of requests per second and their uptime. However DARGOS developers claim that it is easily extensible to new monitoring environments, and to incorporate new metrics.

to maximize their exploitation and minimize their costs. RESERVOIR introduces the Lattice (Clayman et al., 2010a) as a non-intrusive monitoring framework which can be integrated into many monitoring systems. Lattice is an open-source monitoring system natively oriented to infrastructure monitoring and mainly implemented for working in highly dynamic cloud environments including a large number of resources. However, RESERVOIR does not address the issue of directly providing monitoring information to cloud customers. The functionalities of Lattice are focused on the collection and distribution of monitoring data through either multicast or UDP protocol. Therefore, Lattice does not provide functionalities for visualization, evaluation and automated alerting (Katsaros et al., 2011a).

Lattice conceptually offers a design intended to enable application providers to build a monitoring solution fitting their own unique use case in terms of distribution frameworks such as edge computing. However, since a library of monitoring probes to be easily reused in the Lattice platform has not been provided so far, it is not highlighted as a monitoring system within edge computing frameworks.

4.1.9. JCatascopia

JCatascopia (Trihinas et al., 2014), implemented in the Java programming language, is a monitoring tool which is able to monitor federated clouds operating on different cloud providers. It can retrieve heterogeneous monitoring data both at infrastructure level (e.g., memory and CPU) and at application level (e.g., service availability and throughput). JCatascopia is able to adaptively filter measured values of monitored metrics upon small delta differences compared to prior values, in order to reduce the storage and network overhead. Another interesting feature is the possibility to adapt the monitoring activity after machine migration. To this end, each message transmitted to the monitoring server contains the IP address of the monitored resource, so at each change the server is notified. JCatascopia is neither aimed at directly being able to compute cost assessment, nor is it capable of recognizing the topology of a deployed application.

In the SWITCH project, in order to develop a monitoring system, JCatascopia has been chosen as the baseline technology which was extended in this work to fulfil the requirements of containerized applications. Since JCatascopia is written in Java, each container instance which includes a monitoring probe requires some packages and a certain amount of memory for a Java Virtual Machine (JVM) even if the monitored application running alongside the monitoring probe in the container is not programmed by Java. Therefore, containerized monitoring probes in the SWITCH project have been implemented through StatsD protocol[27] available for many programming languages such as C/C++.

4.1.10. Tower 4Clouds[28]

Tower 4Clouds (Miglierina and Nitto, 2016) which is a multi-cloud monitoring platform developed as a part of the MODAClouds project[29] collects metrics at VM and application levels. Self-registering components named data collectors measure metrics and send the monitoring data to a central entity called data analyzer. Tower 4Clouds is a modular platform which allows application providers to build their own monitoring data collectors for custom metrics. It stores the monitoring data in InfluxDB or Graphite, performs analysis on the sorted data and shows results through Web-based user interface tools such as Grafana at runtime. This open-source monitoring platform continuously checks a set of monitoring rules which determine the application health and QoS constraints. If any rule becomes true, an associated action will be triggered such as notifying other components (e.g. self-adapter) by performing REST calls. These rules can be automatically generated through the MODAClouds Integrated Development Environment (IDE) or defined by application designers. This monitoring platform is able to filter monitoring data at various levels of abstraction, handle the heterogeneity of resources being monitored and also autonomously to deal with the scaling actions or live-migration of services from one VM to another one. It should be noted that as a data collector in Tower 4Clouds is written in Java, there is the same problem from which native JCatascopia is suffering: specific packages dependencies and a specific amount of memory are required for a JVM.

However the MODAClouds consortium planned to work towards having Docker container images as another method of obtaining a design-time platform to make cloud applications, they did not extend the Tower 4Clouds in order to be capable of container-level monitoring as well. On the other hand, it is a highly composable open-source platform that helps it to be extended or integrated with containerized and edge computing scenarios.

4.2. Cloud monitoring tools' support for functional requirements

In Table 8, a list of the previously-mentioned cloud monitoring systems and their functional requirements in a general sense is provided. These features are investigated in order to find out an

[27] The StatsD protocol, https://github.com/etsy/statsd/wiki.

[28] Tower 4Clouds, http://deib-polimi.github.io/tower4clouds/docs/.

[29] The MODACLouds project, http://multiclouddevops.com/technologies.html.

Table 9
Detailed analysis of functional requirements for cloud monitoring tools.

Monitoring level	Functional requirements	Zenoss	Ganglia	Zabbix	Nagios	OpenNebula	PCMONS	DARGOS	Lattice	JCatascopia	Tower 4Clouds
Common in all monitoring levels	Usable by the provider and customers in a multi-tenant cloud environment	No[a]	No	Yes	No	Yes	No	Yes	No	No	Yes
	Provide the functionality for visualization	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
	Able to filter measured values to diminish data exchanges	No	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes
	Able to be tuned to any desired monitoring time interval	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Include capability of long-term storing of measured values	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
	Support scaling adaptation policies for large-scale dynamic environments	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes
	Able to set up automated alerts	Yes	No	Yes	Yes	No[f]	Yes	Yes	No	No[g]	Yes
	Support different adaptation actions e.g. service migration, etc.	No	Yes	No	No	Yes	No	No	Yes	Yes	Yes
	Automatic installation and configuration of monitoring system	Yes[b]	Yes/No[c]	Yes[d]	No[e]	Yes	No	Yes	Yes	Yes	Yes
	Able to be customized based on monitoring needs	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Both VM level and container level	Independent from underlying cloud infrastructure provider	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes
	Quickly react to the dynamic resource management changes over time	Yes	Yes	No	No[h]	Yes/No[i]	No	Yes	Yes	Yes	Yes
	Support monitoring of all types of hardware virtualizations	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Offer an API to expose monitoring data	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
End-to-end link quality level	Investigate the whole range of end-to-end network QoS properties	No[j]	No	Yes[l]	Yes[n]	No	No	No	No	No	No
	Support on-demand network configuration	Yes	No	Yes	Yes	No	No	No	No	No	No
	Able to reach the device in spite of filters and firewalls	No[k]	No	No[m]	No	No	No	No	No	No	No
	Consider user's link conditions (e.g. network quality)	No	No	No	No	No	No	No	No	No	No
Application level	Able to deal with the application topology and reconfiguration	Yes[o]	No	No	No	No	No	No	No	No	No
	Define effective measurement interval upon the application performance	No	No	No	No	No	No	No	No	No	No
	Support multi-tier applications	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes
	Able to be adapted with time-varying application adaptation goals	Yes	Yes	Yes	No	No	No	Yes	No[p]	Yes	Yes

[a] Zenoss Service Dynamics is a monitoring platform able to perform multi-tenant operations. However, it is a commercial product.

[b] Auto-discovery in Zenoss should be manually activated. This means it is not able to perform on a periodic basis automatically.

[c] Ganglia lacks the capability of auto-discovery at inter-cluster level.

[d] In order to activate auto-discovery, some configurations need to be manually managed in Zabbix server.

[e] Nagios has enterprise monitoring solutions with auto-discovery that are not free.

[f] OpenNebula has an approach called Hook Manager to trigger custom scripts for changes (e.g. SHUTDOWN, BOOTING, etc.) in state of VMs or hosts. However, the usage of hooks is mainly aimed at having more high availability and strategies on the infrastructure. Moreover, OpenNebula offers the OneFlow component to allow administrators to specify auto-scaling rules based on monitoring metrics.

[g] Automated alerting ability of JCatascopia has been provided. However, this part of the project is proprietary and not publically released.

[h] Nagios is not suitable for monitoring environments that need a high rate of sampling (Katsaros et al., 2011b; Paterson, 2010).

[i] The pull mode usable in OpenNebula is not usable for large-scale environments with high update frequencies. For example, having around 50 VMs, the proper monitoring period would be around 5 minutes. However, the push mode is more scalable for such large clouds.

[j] Fping ZenPack is able to measure latency and packet loss.

[k] Fping ZenPack is a ping-like monitoring tool which uses just ICMP protocol to examine if a target node is responding.

[l] In comparison with Zenoss, Zabbix is able to measure more network related parameters.

[m] However Zabbix supports monitoring by TCP, SNMP and ICMP checks, as well as over IPMI, SSH, JMX, Telnet and also using custom protocols, it lacks automatic network traffic engineering and re-routing.

[n] There exist different Nagios plugins to monitor network download speed, bandwidth usage, network connections, packet loss, etc.

[o] Zenoss Control Center which enables an application to run as a set of distributed services knows where they are running and how they are connected to each other.

[p] A future objective for the Lattice monitoring tool is making the ability to dynamically reprogram the measurement and change the monitoring data source at runtime (Clayman et al., 2010a).

appropriate base-line technology for the needs of monitoring applications deployed based on edge computing framework. Monitoring tools compared in this table are not aimed primarily at monitoring Docker containers; however, some of them have a module or extension to monitor Docker containers. Moreover, in details, Table 9 underlines which functional requirements in the taxonomy described earlier have been addressed by each of cloud monitoring systems from free and open-source software viewpoint.

4.3. Cloud monitoring tools' support for non-functional requirements

Table 10 presents the analysis of the essential non-functional requirements for the previously-mentioned cloud monitoring tools. The goal of the comparison is to specify and trade-off the strengths, drawbacks and challenges which have been encountered in the context of self-adaptive applications in terms of edge computing.

Table 10
Non-functional requirement analysis for cloud monitoring systems.

Tool	Scalability	Robustness	Non-intrusiveness	Interoperability	Live-migration support
Zenoss	Yes	No	Yes	Yes	No
Ganglia	Yes	Yes	Limited[a]	Yes	Yes
Zabbix	Yes	No	Yes	Yes	No
Nagios	No	No	Limited[b]	Yes	No
OpenNebula	Yes	Yes	Yes	No	Limited[c]
PCMONS	No	No	Limited[d]	Yes	No
DARGOS	Yes	No	Yes	No	No
Lattice	Yes	Yes	Yes	Yes	Yes
JCatascopia	Yes	Yes	Limited[e]	Yes	Yes
Tower 4Clouds	Yes	Yes	Limited[f]	Yes	Yes

Note: Comparison in this table is upon the reviewed literature and based on conducting experiments with the tools.
[a] Ganglia daemon called gmond which is running on each monitored node adds an overhead because of both XML event encoding and multicast updates (Arabnejad et al., 2017).
[b] In Nagios, there are numerous service checks which are resource intensive in terms of overhead incurred by notable CPU and IO usage.
[c] Individual configurations in OpenNebula are needed to migrate VMs from one node to another. For example, if the hypervisor is KVM, libvirt's TCP has to be configured. Also a folder should be already shared on both source and destination nodes.
[d] Since PCMONS has been implemented as a module for the Nagios monitoring system and hence both principally act in the same direction.
[e] As a monitoring probe is written in Java, it requires some packages and a certain amount of memory for a JVM.
[f] As a data collector is written in Java, it needs some packages and a specific amount of memory for a JVM.

5. Conclusions and future outlook

Monitoring for adaptive edge computing applications has recently gained a wide range of attention in the context of a "future Internet", as a field that still needs to be fully scrutinized and improved. Since the self-adaptation of applications developed for edge computing frameworks is at an early stage of research and development, we, as the members of cloud community, believe this review paper would serve as an important reference for further research in this field. The paper also highlights several challenges and technical problems in existing cloud monitoring approaches for edge computing purposes, which could be considered to enhance the performance of such adaptive applications.

We have also derived a taxonomy for the main functional and non-functional requirements that cloud monitoring systems should address, and we have related contributions provided in literature so far. More importantly, this review paper has compared several widely used cloud monitoring tools, both open source and commercial, along with their capabilities and shortcomings, as well as how these monitoring systems meet varied requirements. This comprehensive comparison offers companies, which are providing services based on edge computing, an opportunity to gain insights of monitoring tools usable in different but interdependent virtualization layers. However, our comparison shows that an integrated solution that fully monitors all the layers of an edge computing scenario is currently unavailable. Such a solution would need to provide some abstraction layer so that there is at least some commonality in the way these different layers are monitored. Moreover, some requirements have not been really fully met by any of the existing cloud-based monitoring technologies.

Another interesting paradigm is the inclusion of more monitoring systems for container-based virtualization technologies able to provide a lightweight mechanism for initiating, deploying, scaling and moving services between infrastructures within edge computing frameworks. Container management systems such as Kubernetes or OpenShift Origin are lightweight platforms able to orchestrate containers and automatically provide horizontal scalability of applications. However, their native scaling approaches are principally based on CPU usage; no matter for example how workload intensity or application performance is behaving. Therefore, an interesting field of further studies would be the investigation of additional monitoring systems, preferably the ones that are capable of being integrated with container management systems and also published under the Apache 2 license, for easier customization and integration possibilities into other systems.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreements No. 643963 (SWITCH project: Software Workbench for Interactive, Time Critical and Highly self-adaptive cloud applications) and No. 644179 (ENTICE project: dEcentralised repositories for traNsparent and efficienT vIrtual maChine opErations).

References

Aceto, G., Botta, A., Donato, W.D., Pescape, A., 2013. Cloud monitoring: a survey. Comput. Netw. 57 (9), 2093–2115. doi:10.1016/j.comnet.2013.04.001.

Aceto, G., Botta, A., Donato, W.D., Pescape, A., 2012. Cloud Monitoring: definitions, issues and future directions. In: Proceedings of the 1st International Conference on Cloud Networking (CLOUDNET). IEEE, Paris, France, pp. 63–67. doi:10.1109/CloudNet.2012.6.

Agelastos, A., Allan, B., Brandt, J., Cassella, P., Enos, J., Fullop, J., Gentile, A., et al., 2014. The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, New Orleans, USA, pp. 154–165. doi:10.1109/SC.2014.18.

Ahmed, A., Ahmed, E., 2016. A survey on mobile edge computing. In: Proceedings of the 10th IEEE international conference on intelligent systems and control (ISCO 2016). IEEE, Coimbatore, India, pp. 1–8. doi:10.13140/RG.2.1.3254.7925.

Ahmed, E., Rehmani, M.H., 2017. Mobile edge computing: opportunities, solutions, and challenges. Future Gener. Comput. Syst. 70, 59–63.

Al-Hazmi, Y., Campowsky, K., Magedanz, T., 2012. A monitoring system for federated clouds. In: Proceedings of the 1st International Conference on Cloud Networking (CLOUDNET). IEEE, Paris, France, pp. 68–74. doi:10.1109/CloudNet.2012.6483657.

Alcaraz-Calero, J.M., Aguado, J.G., 2015. Comparative analysis of architectures for monitoring cloud computing infrastructures. Future Gener. Comput. Syst. 47, 16–30.

Alhamazani, K., Ranjan, R., Mitra, K., Rabhi, F., Jayaraman, P.P., Khan, S.U., Guabtni, A., Bhatnagar, V., 2015. An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art. Computing 97 (4), 357–377 doi:10.1007.

Arabnejad, V., Bubendorfer, K., Ng, B., 2017. Scheduling deadline constrained scientific workflows on dynamically provisioned cloud resources. Future Gener. Comput. Syst. 75, 348–364. doi:10.1016/j.future.2017.01.002.

Astuto, B.N., Mendonca, M., Nguyen, X.N., Obraczka, K., Turletti, T., 2014. A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Commun. Surv. Tut. 16 (3), 1617–1634.

Beserra, D., Moreno, E.D., Endo, P.T., Barreto, J., 2016. Performance evaluation of a lightweight virtualization solution for HPC I/O scenarios. In: Proceedings of 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, Budapest, Hungary, pp. 4681–4686.

Bonomi, F., Milito, R., Natarajan, P., Zhu, J., 2014. Fog computing: a platform for internet of things and analytics. In: Bessis, N., Dobre, C. (Eds.), Big Data and Internet of Things: A Roadmap For Smart Environments. Springer International Publishing, pp. 169–186.

Caglar, F., Gokhale, A., 2014. iOverbook: intelligent resource-overbooking to support soft real-time applications in the cloud. In: Proceedings of 2014 IEEE 7th international conference on Cloud computing (CLOUD). IEEE, Anchorage, USA, pp. 538–545.

Cervino, J., Rodriguez, P., Trajkovska, I., Mozo, A., Salvachua, J., 2011. Testing a cloud provider network for hybrid P2P and cloud streaming architectures. In: Proceedings of IEEE International Conference on Cloud Computing (CLOUD). IEEE, Washington DC, USA, pp. 356–363. doi:10.1109/CLOUD.2011.52.

Chang, H., Chang, Y., Hsiao, S., 2014. Scalable network file systems with load balancing and fault tolerance for web services. J. Syst. Softw. 93, 102–109.

Chaves, S.A.D., Uriarte, R.B., Westphall, C.B., 2011. Toward an architecture for monitoring private clouds. IEEE Commun. Mag. 49 (12), 130–137. doi:10.1109/MCOM.2011.6094017.

Chen, K., Chang, Y., Hsu, H., Chen, D., Huang, C., Hsu, C., 2014. On the quality of service of cloud gaming systems. IEEE Trans. Multimed. 16 (2), 480–495. doi:10.1109/TMM.2013.2291532.

Clayman, S., Galis, A., Chapman, C., Toffetti, G., Rodero-Merino, L., Vaquero, L.M., Nagin, K., Rochwerger, B., et al., 2010a. Monitoring service clouds in the future. In: Internet, In:, Tselentis, G., et al. (Eds.), Towards the Future Internet. IOS Press, pp. 115–126.

Clayman, S., Galis, A., Mamatas, L., 2010b. Monitoring virtual networks with lattice. In: Proceedings of 2010 IEEE/IFIP Network Operations and Management Symposium Workshops (NOMS Wksps). IEEE, Osaka, Japan, pp. 239–246. doi:10.1109/NOMSW.2010.5486569.

Clayman, S., Toffetti, G., Galis, A., Chapman, C., 2012. Monitoring services in a federated cloud: the reservoir experience. In: Villari, M., Brandic, I., Tusa, F. (Eds.), Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice. IGI Global, pp. 242–265. doi:10.4018/978-1-4666-1631-8.ch013.

Confais, B., Lebre, A., Parrein, B., 2016. Performance analysis of object store systems in a fog/edge computing infrastructures. In: Proceedings of the 8th International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, Luxembourg, pp. 294–301. doi:10.1109/CloudCom.2016.0055.

Desertot, M., Escoffier, C., Donsez, D., 2005. Autonomic management of J2EE edge servers. In: Proceedings of the 3rd international workshop on Middleware for grid computing. ACM, New York, USA, pp. 1–6.

Dusia, A., Yang, Y., Taufer, M., 2015. Network quality of service in Docker containers. In: Proceedings of 2015 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, Chicago, USA, pp. 527–528.

Emeakaroha, V.C., Ferreto, T.C., Netto, M.A.S., Brandic, I., Rose, C.A.F.D., 2012. CASViD: application level monitoring for SLA violation detection in clouds. In: Proceedings of the 36th Annual Computer Software and Applications Conference (COMPSAC). IEEE, Izmir, Turkey, pp. 499–508. doi:10.1109/COMPSAC.2012.68.

Esposito, C., Ficco, M., Palmieri, F., Castiglione, A., 2015. A knowledge-based platform for Big Data analytics based on publish/subscribe services and stream processing. Knowl. Based Syst. 79, 3–17. https://doi.org/10.1016/j.knosys.2014.05.003.

Evans, K., Jones, A., Preece, A., Quevedo, F., Rogers, D., Spasic, I., Taylor, I., Stankovski, V., Taherizadeh, S., Trnkoczy, J., Suciu, G., Suciu, V., Martin, P., Wang, J., Zhao, Z., 2015. Dynamically reconfigurable workflows for time-critical applications. In: Proceedings of International workshop on Workflows in support of large-scale science (WORKS 15). ACM, Austin, USA, pp. 1–10. doi:10.1145/2822332.2822339.

Farokhi, S., Lakew, E.B., Klein, C., Brandic, I., Elmroth, E., 2015. Coordinating CPU and memory elasticity controllers to meet service response time constraints. In: Proceedings of International Conference on Cloud and Autonomic Computing (ICCAC). IEEE, Boston, USA, pp. 69–80. doi:10.1109/ICCAC.2015.20.

Farris, I., Taleb, T., Bagaa, M., Flinck, H., 2017. Optimizing service replication for mobile delay-sensitive applications in 5G edge network. In: Proceedings of 2017 IEEE International Conference on Communications (ICC). Paris, France, pp. 1–6. doi:10.1109/ICC.2017.7997282.

Fatema, K., Emeakaroha, V.C., Healy, P.D., Morrison, J.P., Lynn, T., 2014. A survey of cloud monitoring tools: taxonomy, capabilities and objectives. J. Parallel Distrib. Comput. 74 (10), 2918–2933. doi:10.1016/j.jpdc.2014.06.007.

Fournier, F., Kofman, A., Skarbovsky, I., Skarlatidis, A., 2015. Extending event-driven architecture for proactive systems. In: Proceedings of EDBT/ICDT Workshops. Brussels, Belgium, pp. 104–110.

Garcia-Valls, M., Cucinotta, T., Lu, C., 2014. Challenges in real-time virtualization and predictable cloud computing. J. Syst. Archit. 60 (9), 726–740.

Gorbil, G., Perez, D.G., Cuesta, E.H., 2014. Principles of pervasive cloud monitoring. In: Czachorski, T., Gelenbe, E., Lent, R. (Eds.), Information Sciences and Systems 2014. Springer International Publishing, pp. 117–124. doi:10.1007/978-3-319-09465-6_13.

Grozev, N., Buyya, R., 2013. Performance modelling and simulation of three-tier applications in cloud and multi-cloud environments. Comput. J. 58 (1), 1–22. doi: https://doi.org/10.1093/comjnl/bxt107.

Gupta, U., 2015. Monitoring in IOT enabled devices. Int. J. Adv. Netw. Appl. 7 (1), 2622–2625. arXiv: 1507.03780.

Hazarika, B., Singh, T.J., 2015. Survey paper on cloud computing & cloud monitoring: basics. SSRG Int. J. Comput. Sci. Eng. 2 (1), 10–15 ISSN:2348–8387.

He, Q., Han, J., Yang, Y., Grundy, J., Jin, H., 2012. QoS-driven service selection for multi-tenant SaaS. In: Proceedings of 2012 IEEE 5th international conference on Cloud computing (CLOUD). IEEE, Honolulu, USA, pp. 566–573.

Hsu, W., Lo, C., 2014. QoS/QoE mapping and adjustment model in the cloud-based multimedia infrastructure. IEEE Syst. J. 8 (1), 247–255. doi:10.1109/JSYST.2013.2253035.

Islam, S., Keung, J., Lee, K., Liu, A., 2012. Empirical prediction models for adaptive resource provisioning in the cloud. Future Gener. Comput. Syst. 28 (1), 155–162.

Issariyapat, C., Pongpaibool, P., Mongkolluksame, S., Meesublak, K., 2012. Using Nagios as a groundwork for developing a better network monitoring system. In: Proceedings of PICMET'12 Conference on Technology Management for Emerging Technologies (PICMET). IEEE, Vancouver, Canada, pp. 2771–2777.

Jamshidi, P., Sharifloo, A.M., Pahl, C., Metzger, A., Estrada, G., 2015. Self-learning cloud controllers: fuzzy q-learning for knowledge evolution. In: Proceedings of International Conference on Cloud and Autonomic Computing (ICCAC). IEEE, Boston, USA, pp. 208–211. doi:10.1109/ICCAC.2015.35.

Jesus-Gil, J.D., Botero, J.F., 2016. Network functions virtualization: a survey. IEEE Latin Am. Trans. 14 (2), 983–997.

Katsaros, G., Kubert, R., Gallizo, G., Wang, T., 2011a. Monitoring: a fundamental process to provide QoS guarantees in cloud-based platform. In: Benatallah, B. (Ed.), Cloud Computing: Methodology, System, and Applications. CRC Press, pp. 325–341. doi:10.1201/b11149-18, ISBN: 9781439856413.

Katsaros, G., Kubert, R., Gallizo, G., 2011b. Building a service-oriented monitoring framework with REST and Nagios. In: Proceedings of 2011 IEEE International Conference on Services Computing (SCC). IEEE, Washington DC, USA, pp. 426–431.

Kratzke, N., Quint, P., 2017. Understanding cloud-native applications after 10 years of cloud computing - a systematic mapping study. J. Syst. Softw. 1–16. http://dx.doi.org/10.1016/j.jss.2017.01.001.

Kwon, S., Noh, J., 2013. Implementation of monitoring system for cloud computing. Int. J. Mod. Eng. Res. (IJMER) 3 (4), 1916–1918.

Lampe, U., Wu, Q., Hans, R., Miede, A., Steinmetz, R., 2013. To frag or to be fragged - an empirical assessment of latency in cloud gaming. In: Proceedings of the 3rd International Conference on Cloud Computing and Services Science (CLOSER 2013). Aachen, Germany, pp. 5–12.

Lee, I., Lee, K., 2015. The internet of things (IoT): applications, investments, and challenges for enterprises. Bus. Horiz. 58 (4), 431–440. doi:10.1016/j.bushor.2015.03.008.

Leitner, P., Inzinger, C., Hummer, W., Satzger, B., Dustdar, S., 2012. Application-level performance monitoring of cloud services based on the complex event processing paradigm. In: Proceedings of the 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA'12). IEEE, Taipei, Taiwan, pp. 1–8. doi:10.1109/SOCA.2012.6449437.

Liaqat, M., Ninoriya, S., Shuja, J., Ahmad, R.W., Gani, A., 2016. Virtual machine migration enabled cloud resource management: a challenging task. Distrib. Parallel Clust. Comput. 1–7. https://arxiv.org/pdf/1601.03854.

Massie, M.L., Chun, B.N., Culler, D.E., 2004. The ganglia distributed monitoring system: design, implementation, and experience. Parallel Comput. 30 (7), 817–840. doi: http://dx.doi.org/10.1016/j.parco.2004.04.001.

Mastelic, T., Emeakaroha, V.C., Maurer, M., Brandic, I., 2012. M4Cloud: generic application level monitoring for resource-shared cloud environments. In: Proceedings of the 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012). Springer, Porto, Portugal, pp. 522–532.

Meera, A., Swamynathan, S., 2013. Agent based resource monitoring system in IaaS cloud environment. In: Proceedings of the 1st International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA). Elsevier, Kalyani, India, pp. 200–207.

Miglierina, M., Nitto, E.D., 2016. Monitoring in a multi-cloud environment. In: Nitto, E.D., Matthews, P., Petcu, D., Solberg, A. (Eds.), Model-Driven Development and Operation of Multi-Cloud Applications - The MODAClouds Approach. Springer International Publishing, pp. 47–52.

Minerva, R., Crespi, N., 2017. Technological evolution of the ICT sector. In: Minerva, R., Crespi, N. (Eds.), Networks and New Services: A Complete Story,. Springer International Publishing, pp. 53–87.

Mohamaddiah, M.H., Abdullah, A., Subramaniam, S., Hussin, M., 2014. A survey on resource allocation and monitoring in cloud computing. Int. J. Mach. Learn. Comput. 4 (1), 31–38. doi:10.7763/IJMLC.2014.V4.382.

Mohit, M., 2010. A comprehensive solution to cloud traffic tribulations. Int. J. Web Serv. Comput. 1 (2), 1–13 ISSN:0976-9811.

Mongkolluksamee, S., 2010. Strengths and limitations of Nagios as a network monitoring solution. In: Proceedings of the 7th International Joint Conference on Computer Science and Software Engineering (JCSSE 2010). Bangkok, Thailand, pp. 96–101.

Montero, R.S., Rojas, E., Carrillo, A.A., Llorente, I.M., 2017. Extending the cloud to the network edge. IEEE Comput. 50 (4), 91–95.

Murphy, J.W., 2008. Snoscan: An Iterative Functionality Service Scanner for Large Scale Networks Thesis no.: 1461885. Department of Electrical and Computer Engineering, Iowa State University, Iowa, USA.

Paterson, M., 2010. Evaluation of Nagios for Real-Time Cloud Virtual Machine Monitoring. University of Victoria, Canada.

Povedano-Molina, J., Lopez-Vega, J.M., Lopez-Soler, J.M., Corradi, A., Foschini, L, 2013. DARGOS: a highly adaptable and scalable monitoring architecture for multi-tenant Clouds. Future Gener. Comput. Syst. 29 (8), 2041–2056.

Preeth, E.N., Mulerickal, F.J.P., Paul, B., Sastri, Y., 2015. Evaluation of Docker containers based on hardware utilization. In: Proceedings of 2015 International Conference on Control Communication & Computing India (ICCC). IEEE, Kerala, India, pp. 697–700.

Rao, J., Bu, X., Xu, C., Wang, K., 2011. A distributed self-learning approach for elastic provisioning of virtualized cloud resources. In: Proceedings of 2011 IEEE 19th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE, Singapore, pp. 45–54.

Rossi, F., Oliveira, I., de-Rose, C., Calheiros, R., Buyya, R., 2015. Non-invasive estimation of cloud applications performance via hypervisor's operating systems counters. In: Proceedings of the 14th International Conference on Networks (ICN). NexComm, Barcelona, Spain, pp. 177–184.

Samimi, F.A., McKinley, P.K., Sadjadi, S.M., Tang, C., Shapiro, J.K., Zhou, Z., 2007. Service clouds: distributed infrastructure for adaptive communication services. IEEE Trans. Netw. Serv. Manag. 4 (2), 84–95.

Satyanarayanan, M., 2017. The emergence of edge computing. Computer 50 (1), 30–39.

Seo, K., Hwang, H., Moon, I., Kwon, O., Kim, B., 2014. Performance comparison analysis of Linux container and virtual machine for building cloud. Adv. Sci. Technol. Lett. 66, 105–111. doi:10.14257/astl.2014.66.25.

Shao, J., Wang, Q., 2011. A performance guarantee approach for cloud applications based on monitoring. In: Proceedings of the IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW). IEEE, Munich, Germany, pp. 25–30. doi:10.1109/COMPSACW.2011.15.

Sharma, V., Srinivasan, K., Jayakody, D.N.K., Rana, O., Kumar, R., 2017. Managing service-heterogeneity using osmotic computing. arXiv preprint, 1–7, https://arxiv.org/pdf/1704.04213.pdf.

Shi, W., Dustdar, S., 2016. The promise of edge computing. Computer 49 (5), 78–81.

Simmonds, E., Harrington, J., 2009. SCF/FEF Evaluation of Nagios and Zabbix Monitoring Systems. SCF/FEF, pp. 1–9.

Stankovski, V., Trnkoczy, J., Taherizadeh, S., Cigale, M., 2016. Implementing time-critical functionalities with a distributed adaptive container architecture. In: Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services (iiWAS2016). ACM, Singapore, pp. 455–459. doi:10.1145/3011141.3011202.

Sugapriya, K., Jeya, J.S., 2015. A survey on enhanced scheme for multiple cloud resource matchmaking using trust aware framework. Int. J. Adv. Res. Comput. Sci. Softw. Eng. 5 (11), 1–4 ISSN:2277-128X.

Tader, P., 2010. Server monitoring with Zabbix. Linux J. (195).

Taherizadeh, S., Taylor, I., Jones, A., Zhao, Z., Stankovski, V., 2016a. A network edge monitoring approach for real-time data streaming applications. In: Proceedings of the 13th International Conference on Economics of Grids, Clouds, Systems and Services (GECON 2016). ACM, Athens, Greece, pp. 1–12.

Taherizadeh, S., Jones, A.C., Taylor, I., Zhao, Z., Martin, P., Stankovski, V., 2016b. Runtime network-level monitoring framework in the adaptation of distributed time-critical cloud applications. In: Proceedings of the 22nd International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'16). Las Vegas, USA, pp. 78–83.

Telesca, A., Grigore, A., Delort, C., Fuchs, U., Haller, B.V., Denes, E., Vyvre, P.V., et al., 2014. System performance monitoring of the ALICE data acquisition system with Zabbix. J. Phys. 513 (6), 1–7.

Toosi, A.N., Calheiros, R.N., Buyya, R., 2014. Interconnected cloud computing environments: challenges, taxonomy, and survey. ACM Comput. Surv. 47 (1). doi:10.1145/2593512.

Trihinas, D., Pallis, G., Dikaiakos, M.D., 2014. JCatascopia: monitoring elastically adaptive applications in the cloud. In: Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). IEEE, Chicago, USA, pp. 226–235.

Vangeepuram, R.T., 2016. Performance Comparison of Cassandra in LXC and Bare metal Thesis no.: MSEE-2016:38. Faculty of Computing, Blekinge Institute of Technology.

Verma, A., Bhardwaj, N., 2016. A review on routing information protocol (RIP) and open shortest path first (OSPF) routing protocol. Int. J. Future Gener. Commun. Netw. 9 (4), 161–170.

Villari, M., Fazio, M., Dustdar, S., Rana, O., Ranjan, R., 2016. Osmotic computing: a new paradigm for edge/cloud integration. IEEE Cloud Comput. 3 (6), 76–83.

Wamser, F., Loh, F., Seufert, M., Tran-Gia, P., Bruschi, R., Lago, P., 2017. Dynamic cloud service placement for live video streaming with a remote-controlled drone. In: Proceedings of the 15th IFIP/IEEE International Symposium on Integrated Network Management (IM) (Demonstration). IEEE, Lisbon, Portugal.

Ward, J.S., Barker, A., 2014. Observing the clouds: a survey and taxonomy of cloud monitoring. J. Cloud Comput. 3 (24), 1–30.

Wood, T., Cherkasova, L., Ozonat, K., Shenoy, P., 2008. Profiling and modeling resource usage of virtualized applications. In: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware'08. Springer, Berlin, Germany, pp. 366–387.

Xiao, Z., Liang, P., Tong, Z., Li, K., Khan, S.U., Li, K., 2016. Self-adaptation and mutual adaptation for distributed scheduling in benevolent clouds. Concurr. Computat. 29 (5), 1–12. doi:10.1002/cpe.3939.

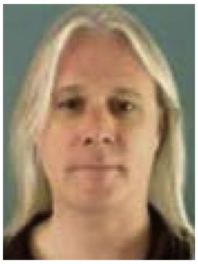
Xiong, P., Pu, C., Zhu, X., Griffith, R., 2013. vPerfGuard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments. In: Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE '13). ACM, New York, USA, pp. 271–282. doi:10.1145/2479871.2479909.

Xu, X., Chen, Y., Alcaraz-Calero, J.M., 2016. Distributed decentralized collaborative monitoring architecture for cloud infrastructures. Cluster Comput. 1–13. doi:10.1007/s10586-016-0675-5.

Zhao, Z., Martin, P., Wang, J., Taal, A., Jones, A., Taylor, I., Stankovski, V., Vega, I.G., Suciu, G., Ulisses, A., de-Laat, C., 2015. Developing and operating time critical applications in clouds: the state of the art and the SWITCH approach. In: Proceedings of the 1st International Conference on Cloud Forward: From Distributed to Complete Computing. Elsevier, Pisa, Italy, pp. 17–28.

Salman Taherizadeh is currently a PhD candidate at the Faculty of Computer and Information Science, University of Ljubljana. Salman earned his Master degree in Information Technology Engineering. He graduated as the MSc first-ranked student of Information Technology Engineering Department, Faculty of Engineering. He is also a researcher at the Faculty of Civil and Geodetic Engineering, University of Ljubljana. He has been working and taking active part in two European projects called SWITCH and ENTICE. His research is focused on methods and technologies for monitoring highly-adaptive time-critical cloud and edge computing applications.

Andrew C. Jones is a senior lecturer in the School of Computer Science & Informatics at Cardiff University. He holds a PhD in Computer Science from CU and his research areas include biodiversity informatics, distributed workflows and Cloud computing. Jones has been an investigator on a number of EC- and UK Research Councils funded Biodiversity Informatics projects (EC projects include i4Life, EU-Brazil OpenBio, BioVeL, 4D4Life, EuroCat, ENBI; UK Research Councils projects include Biodiversity World, SPICE and LITCHI). He also participated in the SHIWA project. He is currently taking part in the H2020 SWITCH project in software engineering for big data.



Ian Taylor is an adjunct research associate professor at the CRC, Notre Dame, and a Reader in Cardiff University, UK. He also consults often to the Naval Research Lab (NRL) and has led the IT development and infrastructures for several startups and redevelopment projects for existing businesses. Ian has a degree in Computing Science and a Ph.D. researching and implementing artificial-neural-network types for the determination of musical pitch. After his Ph.D., he joined the gravitational-wave group at Cardiff where he designed, procured and engineered the implementation of the data acquisition system for the GEO 600 gravitational wave detector. He also wrote the Triana workflow system and managed it thereafter. Ian's research over the last 25 years has covered a broad range of distributed computing areas but he now specializes in Web interaction and APIs, big data applications, open data access, distributed scientific workflows and data distribution, with application areas ranging from audio, astrophysics and engineering to bioinformatics and healthcare. He has managed over 15 research and industrial projects, published over 150 papers, 3 books, acted as guest editor for several special issues in journals and chairs the WORKS Workflow workshop yearly at Supercomputing.

Zhiming Zhao obtained his Ph.D. in computer science in 2004 from University of Amsterdam (UvA). He is currently a senior researcher in the System and Network Engineering group at UvA. He coordinates research effort on quality critical systems on programmable infrastructures in the context of European H2020 projects of SWTICH and ENVRIPLUS. His research interests include software defined networking, workflow management systems, multi agent system and big data research infrastructures.

Vlado Stankovski is an Associate Professor of Computer Science focusing on Distributed, Grid, Cloud and Edge Computing, employed at the Faculty of Civil and Geodetic Engineering, University of Ljubljana. He has been the technical manager of the FP6 DataMiningGrid project, one of the managers of the FP6 InteliGrid project and took part in the FP7 mOSAIC Cloud project. He is currently taking part in two H2020 SWITCH and ENTICE projects in software engineering for big data and advanced cloud computing.