Quality of Service Assurance for Internet of Things Time-Critical Cloud Applications

Experience with the SWITCH and ENTICE projects

Salman Taherizadeh and Vlado Stankovski
University of Ljubljana
Ljubljana, Slovenia
{Salman.Taherizadeh, Vlado.Stankovski}@fgg.uni-lj.si

Abstract—**Various Internet of Things (IoT) applications, such as home automation and disaster early warning systems, are being introduced in various areas of human life and business. Today, a common method for delivery of such applications is via component-based software engineering disciplines based on cloud computing technologies such as containers. However, there are still numerous technological challenges to be solved particularly related to the time-critical Quality of Service (QoS) aspects of such applications. Runtime variations in the workload intensity as the amount of service tasks to be processed may radically affect the application performance perceived by the end-users or lead to the underutilization of resources. In order to assure the QoS of these containerized applications, monitoring is required at both container and application levels. Currently, there is a great lack of such multi-level monitoring systems. In this study, we present an architecture and implementation of a multi-level monitoring framework to ensure system health and adapt an IoT application in response to varying quantity, size and computational requirements of arrival requests. In this work, cloud application adaptation possibility includes horizontal scaling of container-based application instances.**

Keywords—Internet of Things; monitoring; adaptation; cloud.

I. INTRODUCTION

The IoT is a paradigm where things/objects/sensors have a pervasive presence in the Internet. In recent years, IoT systems such as home automation, gaming or early warning systems have emerged as cloud-based services which are increasingly widely used and important, especially to organizations that tend to look beyond the traditional approach of safety applications. As IoT services can be virtualized, replicated and distributed in different cloud infrastructures, cloud computing has become a preferable solution for providing such applications on the Internet. The cloud computing model is a pay-per-use on-demand offer through which organizations can exploit elastic cloud resources and a federated cloud environment to support the QoS needed for running these types of applications. The ultimate goal is to enhance the Quality of Experience (QoE) of their end-users.

Nowadays, a popular cloud technology for the delivery of these applications is through the use of containers (e.g. Docker [1], CoreOS [2], etc.). Due to the lightweight nature of containers and their fast boot time, it is possible to deploy IoT cloud service instances in hosting environments faster and more efficiently than using Virtual Machines (VMs) [1].

Ensuring IoT applications to be able to offer favorable performance has been a challenging issue due to runtime variations in the execution environment such as increase or decrease in the number of connected IoT sensors. Accordingly, the next generation of IoT systems should be built as being self-adaptive, that means, the IoT applications should operate without human intervention [2]. Such applications should be able to detect runtime environmental changes in terms of varied workload intensity as the number of IoT sensors, and then determine their own way of reacting to such changes.

This research work presents a multi-level monitoring approach based upon a non-intrusive design intended to enable IoT-based time-critical cloud-based applications to autonomously reconfigure and adapt to changing workload at runtime. To adapt these applications to the changing conditions, this work presents a rule-based horizontal scaling method to dynamically estimate the number of running containers needed to provide the service. This innovative horizontal scaling method is able to add more container instances into the pool of resources in order to share the workload or remove some running containers, if this does not significantly affect the QoS. The results of our evaluation show that our new adaptation method offers a high level of merits with regard to the automatic scalability of vertualized IoT application components without both resource over and under-provisioning.

The rest of the paper is organized as follows. Section 2 presents the basic framework of an IoT time-critical cloud application. Section 3 describes monitoring requirements as the QoS assurance. Section 4 discusses the implementation of our proposed multi-level monitoring framework. Section 5 presents the architecture of our adaptation solution, followed by empirical evaluation results and finally conclusion respectively in Sections 6 and 7.

[1] Docker, https://www.docker.com/
[2] CoreOS, https://coreos.com/

II. BASIC FRAMEWORK OF AN IOT CLOUD APPLICATION

A. *Use Case*

One typical example for IoT systems is a disaster early warning system. Such systems are developed to provide alerts in a community before disaster occurs. Fig. 1 depicts the basic architecture of an IoT time-critical cloud application including different application components: Call Operator (dedicated and ad-hoc agents), Contact Centre Server (Apache Web server), Database Server (Apache Cassandra server), IP Gateway (e.g. TA900e or Cisco-ASA) and IoT Sensors (transmitters for temperature, barometric pressure, humidity and other environmental variables).

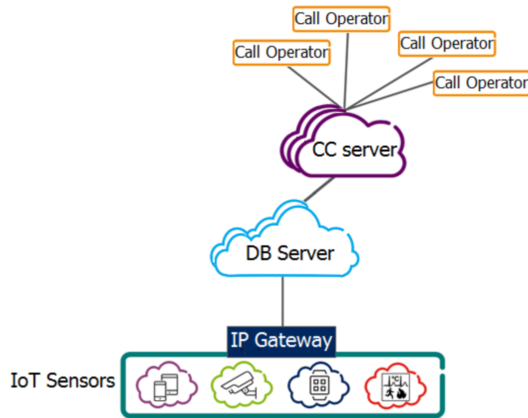

Fig. 1. Example of an Internet of Things (IoT) system.

The basic framework of an IoT cloud application, shown in Fig. 1, consists of the following components:

- IoT Sensors: IoT Sensors can measure different parameters such as water pressure, noise, luminance and other environmental variables.
- IP Gateway: The IP Gateway is a node that allows communication between networks. It receives data over direct radio link or GSM/GPRS from IoT sensors, aggregates the data and sends the data to the database.
- Database Server (DB Server): The DB Server is the database which is used for storing and handling sensed values indexed by time.
- Contact Centre Server (CC Server): The CC server checks sensed data stored in the DB Server and statistics in real-time and sends notifications (such as e-mail, SMS or voice call via SIP-based IP telephony or ordinary PSTN) to Call Operators if values are outside of predetermined thresholds.
- Call Operator: The Call Operators decide whether or not to send an alert to emergency systems or to the public entities.

IoT Sensors and IP Gateway cannot be virtualized as these components have physical items like attached antennas. In this research work, the CC Server and the DB Server can be containerized, replicated and distributed in a federated cloud environment.

III. QUALITY OF SERVICE ASSURANCE

One of the main requirements of containerized self-adaptive early warning applications is to implement a multi-level monitoring tool. This multi-level monitoring tool should be able to monitor execution environment where containerized application components are running on cloud infrastructures. Our implemented monitoring tool considers container-related and application-level parameters.

A. *Container-level monitoring*

Today, cloud computing is realized through the use of VMs or containers. VM-based virtualization is achieved through the use of a hypervisor. The hypervisor emulates machine hardware and then instantiates other VMs along with guest Operating Systems (Guest OSs) on top of that hardware. Each VM instance has a set of its own libraries and software components, and operates within the emulated environment provided by the hypervisor.

On the other hand, containers offer a more modern lightweight approach than VM-based virtualization. A container-based system provides a shared, virtualized OS image consisting of a root file system and a safely shared set of system libraries and executables. This eliminates the need for the use of a hypervisor. Compared to a VM-based system, the use of containers which does not require an OS to boot up as another form of server virtualization is rapidly increasing in popularity. The container-level monitoring tool is able to measure and display runtime value of key attributes (e.g. CPU or memory usage) for a given container such as a containerized CC Server instance.

B. *Application-level monitoring*

Service or application-level monitoring systems measure metrics that present information about the situation of the cloud-based service and its performance. However, although a large number of research works consider the reliability of the underlying cloud infrastructures, there still exists an absence of efficient application-level monitoring techniques to be able to detect and monitor QoS degradation of cloud applications. Monitoring of application-level metrics needs to be done on the application layer. Application-level metrics can be monitored by application-level monitoring probes. The probe could represent a standalone application that runs on the application layer amongst other applications. On the other hand, the application-level probe could be implemented by changing the source code of the application. Also, there are specific service-level metrics which cannot be measured if an application does not provide an interface such as an API for it. In this work, for example, an application-level metric in the conducted use case can be the average response time of the CC Server.

IV. MONITORING FRAMEWORK

A. *Architecture of the monitoring system*

In order to develop a monitoring system to measure metrics, JCatascopia [4] has been chosen as baseline technology which was extended in this work to fulfil the requirements of (1) containerized, (2) self-adaptive, (3) IoT, (4) time-critical cloud-based applications.

Our proposed monitoring system uses an agent-based client-server approach, which is able to support a fully interoperable, highly scalable and light-weight architecture. The distributed nature of this monitoring framework quenches the runtime overhead of system to a number of Monitoring Agents running across different cloud resources. This monitoring system offers a framework to measure, store and report monitoring metrics from different layers e.g. containers as well as possible performance metrics from deployed applications. Fig. 2 shows an overview for the architecture of the proposed monitoring framework.

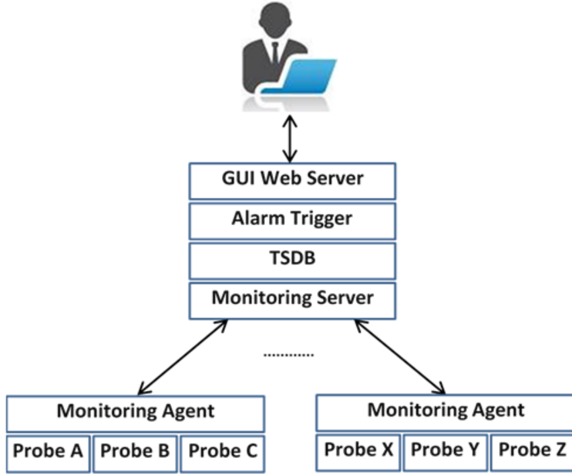

Fig. 2. Architecture of the monitoring system.

The architecture of our designed monitoring framework includes different components namely Monitoring Probes, Monitoring Agents, Monitoring Server, Time Series Database (TSDB), Alarm-Trigger and Graphical User Interface (GUI) Web Server.

1) Monitoring Probes

Monitoring Probes are the actual components that collect individual metrics at different levels such as container and application. For example, a container-level Monitoring Probe can be the component to measure the CPU utilization of a given container e.g. a containerized CC Server instance. Another one can be able to measure the memory percentage usage of the container. Or an application-level Monitoring Probe can monitor the response time of an application running inside the deployed containers. In essence, Monitoring Probes are in charge of gathering values of measured metrics, which are then aggregated by an associated Monitoring Agent.

2) Monitoring Agents

The Monitoring Agent is responsible for the management of metrics collection on a particular element. It aggregates the values measured by Monitoring Probes and then distributes them to the Monitoring Server.

3) Monitoring Server

The Monitoring Server is a component that receives measured metrics from the Monitoring Agents. The collected metrics are then processed and stored in the monitoring TSDB to manage huge amount of structured data.

4) Time Series Database (TSDB)

The monitoring data streams coming from Monitoring Probes/Agents are stored in the TSDB, which is a special database customized for the storage of series of data points. The reason to use the TSDB is the capability of storing huge volumes of time-ordered data more efficiently than it could be stored in a Knowledge Base.

5) Alarm-Trigger

The Alarm-Trigger is a configurable surveillance component which investigates the incoming measured values to initiate actions when irregular incidents occur. This component comprises different thresholds for all monitoring metrics. It notifies the Self-Adapter when the monitoring data reach or exceed a pre-determined threshold level. The Alarm-Trigger is using rule-based mechanism to avoid the complexity of our proposed self-adaptation approach and to prohibit human interventions.

6) GUI Web Server

The GUI Web Server allows all external entities to access the monitoring information stored in the TSDB in a unified way, via prepared REST-based Web services and APIs.

B. Operation of the monitoring system

A Monitoring Agent which is running alongside an application service in a container aggregates the measured values and then transmits them to the Monitoring Server.

The Alarm-Trigger will detect the key quality attributes such as the need for less or more resources on the DB Server or the CC Server, and then the adaptation part dynamically tunes the execution of the whole application to improve the possible performance drops. In order to measure the status of containerized CC Server and DB Server instances, the needed monitoring metrics could be divided in two main categories including container-level metrics and application-level metrics.

V. Runtime Adaptation Mechanism

The proposed runtime adaptation mechanism, shown in Fig. 3, includes various entities when the application executes.

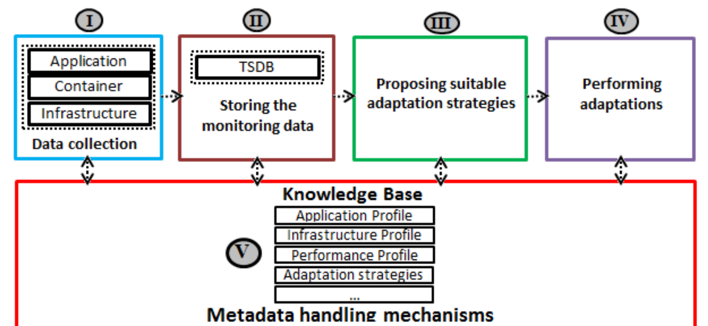

Fig. 3. The proposed runtime adaptation mechanism

In order to make an effective improvement in the performance of IoT time-critical cloud applications, the entities (shown in Fig. 3) will proceed as follows:

I) Data collection

The purpose of Monitoring Probes/Agents is to collect the data that represents the current state of managed elements namely application and container, and then aggregate and transfer the measured values to the Monitoring Server and the Alarm-Trigger. The monitored metrics depend on the use case since the important parameters for each application are different. The Monitoring Probes/Agents should be non-intrusiveness [5], scalable [6], robust [7], interoperable [8] and able to support live-migration [9] as the essential non-functional monitoring requirements needed to support dynamic adaptation of cloud-based applications.

II) Storing the monitoring data

The Monitoring Server receives the collected data and stores it in a TSDB to build a focused and comprehensive representation of the system state. The TSDB can be implemented by the Apache Cassandra technology which is a distributed storage system for managing very large amount of time-ordered data [10]. Concurrently, the Alarm-Trigger investigates if the measured values of monitored parameters exceed predefined limits. In other words, The Alarm-Trigger is a rule-based component which processes the incoming monitoring data streams and notifies the Self-Adapter when predefined thresholds are violated. The Monitoring Server and the Alarm-Trigger should be tightly coupled, i.e. running on the same machine in order to save network bandwidth and computational resources needed for data distribution and processing.

III) Proposing suitable adaptation strategies

When problems are detected, the Self-Adapter is invoked to propose suitable adaptation strategies in terms of increment or decrement in the number of containerized CC Server and DB Server instances. The Self-Adapter is able to automatically identify metrics (e.g. CPU or memory utilization) that are the most predictive for the application performance. The Self-Adapter specifies a set of adaptation actions for the Control-Agent that allows the passage of the whole system from a current state to a desired state. In other words, the Self-Adapter reasons about adaptation changes which should be done to adapt the system to the desired behavior. In this work, adaptation possibility can be horizontal scaling of the DB Server and the CC server.

Besides that, ensuring that these types of IoT applications are able to offer favorable service quality has been a challenging issue due to runtime variations in network conditions intrinsic to connections between individual application components in different tiers. In this case, the Self-Adapter can provide a solution to replicate application components in different cloud infrastructures in order to increase availability and reliability under various network conditions and varied amount of traffic. Therefore, the adaptation action can dynamically connect each component instance to the best possible component instance in each different tier, together offering fully-qualified network performance that was proposed in our previous work [11]. Examples of different application tiers include the application components (the CC Server instances and the DB Server instances) that can run in data centres, gateways/routers that can run in edge devices, or Raspberry Pis and fog devices such as smartphones and automatically driven cars.

IV) Performing adaptations

The Control-Agent which has the full control of application configurations and infrastructure resources e.g. containers finally carries out the adaptation actions defined by the Self-Adapter. This entity is able to increase or decrease the required number of containerized application components on demand even in different cloud data centers that is often an essential requirement for providers of IoT early warning applications running on the cloud.

V) Metadata handling mechanisms

The Knowledge Base will be used to store all information about the current system metadata, awareness and application configuration for analysis, reuse, reasoning, optimization and refinement of design, topology and execution. The knowledge stored in this element describes profiles of all entities (e.g. application profile, infrastructure profile, performance profile, adaptation strategies, etc.), and it is used to interpret monitoring data [12].

VI. RESULTS

We conducted a set of proof-of-concept experiments. Their goal was to examine the design details of the proposed QoS assurance system and to explore the horizontal-scaling adaptation possibility.

The initial set of experiments measured the CC Server's performance. To this end, incoming requests have been generated by the *httperf* tool and sent to the CC Server. The *httperf* tool provides a flexible facility for generating various workload patterns.

For resource intensive applications such as the CC Server, a performance bottleneck could be the CPU power consumption and the memory capacity utilization. In this situation, when the workload density is rising, a possible adaptation mechanism could be horizontal scaling, which can be achieved by adding more running container instances into the pool of resources. This pool of running containers is then able to handle more requests. Another scaling possibility is to stop some of the container instances, if they are not required to avoid resource over-provisioning.

Based on our experiments, the period of time taken to launch a container instance is two seconds. Also, after the container start-up, registering the associated Monitoring Agent in the Monitoring Server takes four seconds. The monitoring interval should be set longer than the container instance's initiation time. In this way, the whole system is able to continue operating properly without losing control over running container instances. Therefore, to prevent any problem at runtime, the monitoring interval has been set to 20 seconds in the experiments.

In order to develop the self-adaptation mechanism, a threshold for every single monitoring metric in different levels has been defined. The Alarm-Trigger is responsible for periodically checking the incoming monitoring data streams and notifies the Self-Adapter when predefined thresholds for metrics are violated. For example, the thresholds for average CPU usage and average memory usage for each Dockerized component (e.g. CC Server) at the container level have been considered to be 80 percent. Moreover, we assume that if the average response time at the application level for the CC Server component is less than 15ms, there is no performance issue and hence, the threshold for average response time of the CC Server has been set to the value of 15ms. A big value for this threshold makes the adaptation method less sensitive to the application performance and more dependent on the infrastructure utilization. However, a very low threshold for the average response time may compel the adaptation method to unnecessarily change the number of container instances whereas the system is currently able to provide users an appropriate performance without any threat.

The final rule for this scenario can be specified as follows. If one of the monitored metrics (average CPU usage, average memory usage or average response time) pertaining to a specific application component (here, for the CC Server) exceeds associated thresholds, the Alarm-Trigger sends an announcement to the Self-Adapter. The Self-Adapter then helps to estimate the number of needed running container instances providing the service since the number of container instances is needed to be increased on demand. Following is a pseudocode of an algorithm which estimates the number of needed containers to be added to a cluster for a certain service (e.g. CC Server) upon metric values measured at container level.

$Increment \leftarrow 0$;
do {
    $Increment \leftarrow Increment + 1$;
    $Expected_{metric} \leftarrow [(Container_{no} * Usage_{metric})/(Container_{no} + Increment)]$;
} while ($Expected_{metric} > Threshold_{metric}$);

In this algorithm, the metric from the actual experiment can be the average CPU and memory usage of all running container instances for a certain service, $Threshold_{metric}$ is the threshold defined for the metric (in our experiment, 80%), $Usage_{metric}$ is the current value of the metric, $Container_{no}$ is the current number of running container instances together providing the service, *Increment* is the number of containers to be added for the service and $Expected_{metric}$ is the expected value of the metric after initiating new container instances.

Both thresholds for average CPU and memory utilization of the cluster which includes the CC Server container instances are considered 80%. This value gives the adaptation method a chance to react to runtime variations in the workload before a performance issue arises. If the workload trend is very even and predictable, these two thresholds can be pushed a little higher than 80%. However, a small value for these thresholds may lead to the over-provisioning problem which wastes costly resources.

In contrast, if the workload density drops at runtime, unnecessary running container instances should be possibly terminated to avoid resource over-provisioning. Based on our proposed conservative strategy, at most one container could be stopped in each adaptation interval in order to make sure that the system offers favorable service quality to end-users. In this way, the system certainly provides acceptable responses upon uncertain environments at runtime. The following algorithm evaluates if one of running container instances can be terminated without any application performance degradation.

$Decrement \leftarrow 0$;
$Expected_{metric} \leftarrow [(Container_{no} * Usage_{metric})/(Container_{no} - 1)]$;
if ($Expected_{metric} < Threshold_{metric}$) then $Decrement \leftarrow 1$;

According to $Expected_{metric}$ (expected value of the metric after the termination of a container instance), value of *Decrement* determines if it is needed to decrease the number of containers running in the cluster.

As shown in Fig. 4, sometimes the workload pattern is slowly rising or falling. Occasionally, it is drastically changing or, on the other hand, gently shaking. If the number of requests is increasing whereas one or more predefined thresholds are reached, it is required to share the workload among more running container instances, thus new containers need to be initiated during the increasing workload. Or, if the workload is decreasing, it is needed to possibly stop unnecessary containers without any QoS degradation perceived by the users. Fig. 4 shows that our proposed multi-level monitoring is able to properly support self-adaptive IoT time-critical cloud-based applications to handle the varying workload by increasing or decreasing the number of running container instances in a service cluster. During this experiment, the average response time of the CC Server was 148ms at the worst case which is quite suitable to address QoS of the application and QoE for the end-users.

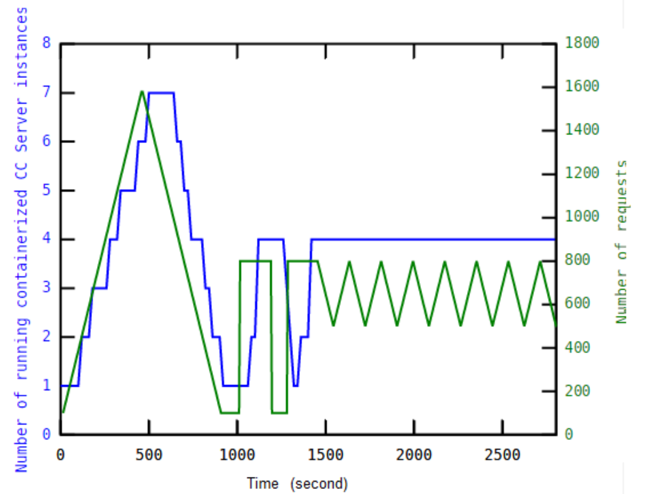

Fig. 4. Number of container instances vs the changing number of requests.

Fig. 4 shows a delay which is a time difference between workload colored green and the number of containers colored blue. This delay implies the existing monitoring interval and the time during which the adaptation action takes place in regard to the changing workload.

VII. CONCLUSION AND DISCUSSION

The Next Generation Internet will increasingly rely on IoT applications. These may include gaming, home automation, supporting infrastructure for robots and disaster early warning systems. These are applications practically covering all of human life and business activities. Software engineers today already prefer to use cloud computing technologies and tools, such as Juju or Fabric8, to build their containerized applications. In this paper, the time-critical QoS aspects of such cloud applications are investigated in terms of varying workload at runtime. It is shown that continuous monitoring of the QoS is required at both container and application levels in order to adapt the application performance to changing workload intensity. Currently, there is a great lack of adequate monitoring systems for this purpose, and this study proposes a non-intrusive multi-level monitoring method.

The conducted experiments have demonstrated the benefits of our presented adaptation approach which helps application providers to avoid under-provisioning as well as over-provisioning of resources in order to prevent QoS degradation and cost overruns at execution time.

We have begun extending our proposed method towards using network-level QoS metrics [13] in a multi-instance architecture. This architecture applies one application instance per one user or one type of users. In this model, any autonomous adaptation mechanism would need to consider more sophisticated options, such as setting up a new monitoring environment for a different type of application instance, which will add to the complexity of the adaptation process for the application.

Our work is included in the software solutions of two ongoing European Horizon 2020 projects: SWITCH [3] and ENTICE [4]. The SWITCH project is funded under the programme for software engineering for IoT and Big Data, while the ENTICE project is funded under the programme of advanced cloud computing. The SWITCH project provides an interactive environment for developing applications and controlling their execution, a real-time infrastructure planner for deploying applications in clouds, and an autonomous system adaptation platform for monitoring and adapting system behaviour. The ENTICE project develops a technology for a federated repository of VM and container images. For this repository, the resource usage, speed, elasticity, redundancy, fault tolerance and other QoS metrics are desired to be considered. In this project, the develped monitoring solution can be used for example to monitor point-to-point network quality among storages in this federated repository.

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

[4] The ENTICE project, http://www.entice-project.eu/

ACKNOWLEDGMENT

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

[1] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder, "Docker containers across multiple clouds and data centers," In: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), 2015, pp. 368–371.

[2] M. Koprivica, 2013. "Self-adaptive requirements-aware intelligent things," International Journal of Internet of Things 2, 1, 2013, 4 pages. DOI:10.5923/j.ijit.20130201.01

[3] A. Botta, W. de-Donato, V. Persico, and A. Pescape, "Integration of cloud computing and internet of things: a survey," Future Generation Computer Systems 56, 2016, pp. 684-700. DOI:10.1016/j.future.2015.09.021

[4] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "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, 2014, pp. 226-235.

[5] S. Suneja, C. Isci, V. Bala, E. De-Lara, and T. Mummert, "Non-intrusive, out-of-band and out-of-the-box systems monitoring in the cloud," In: ACM SIGMETRICS Performance Evaluation Review 42 (1), 2014, pp. 249-261.

[6] S. Clayman, A. Galis, and L. Mamatas, "Monitoring virtual networks with Lattice," In Proceedings of 2011 IEEE/IFIP Network Operations and Management Symposium Workshops (NOMS Wksps), IEEE, 2011, pp. 239-246.

[7] K. Fatema, V. C. Emeakaroha, P. D. Healy, J. P. Morrison, and T. Lynn, "A survey of cloud monitoring tools: Taxonomy, capabilities and objectives," Journal of Parallel and Distributed Computing 74, 10, October 2014, pp. 2918-2933.

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

[9] A. Nadjaran-Toosi, R. N. Calheiros, and R. Buyya, "Interconnected cloud computing environments: Challenges, taxonomy, and survey," ACM Computing Surveys (CSUR) 47 (1), 2014, 47 pages.

[10] D. Namiot, "Time Series Databases," In Proceedings of the XVII International Conference Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL'2015), Russia, 2015, pp. 132-137.

[11] S. Taherizadeh, A. Jones, I. Taylor, Z. Zhao, P. Martin, and V.Stankovski, "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), ACM, Las Vegas, USA, 2016.

[12] F. Zablith, G. Antoniou, M. d'Aquin, G. Flouris, H. Kondylakis, E. Motta, D. Plexousakis, and M. Sabou, "Ontology evolution: a process-centric survey," The Knowledge Engineering Review 30 (01), 2015, pp. 45-75.

[13] S. Taherizadeh, I. Taylor, A. Jones, Z. Zhao, and V. Stankovski, "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, 2016.