U.P.B. Sci. Bull., Series C, Vol. 83, Iss. 4, 2021 ISSN 2286-3540

EFFICIENCY OF APPLICATION MONITORING IN IMPROVING MOBILE COMMUNICATION NETWORKS' RESILIENCE – A CASE STUDY

Viviana Laetitia MINEA[1], Marius MINEA[2], Augustin SEMENESCU[3]

The increasing importance of communications in the era of Internet of Things needs to be supported by robust, efficient, and fully operational communication networks and associated services. This paper presents a comparative study on operational efficiency, resources management and speed of recovery after a failure between two cases, namely running network applications on virtual machines, versus employing Docker containers. Four instances of one application installed virtually on these supports are tested for 48 hours and the results analyzed. Finally, a future development solution with over-imposed artificial intelligence for improved resilience is proposed.

Keywords: Application monitoring, Virtual machines, Docker, software containers, preventive automated maintenance

1. Introduction

Presently, the rapid evolution of smart devices, Internet of Things, smart mobility, smart cities, or other modern intelligent appliances involve increasing mobile communications demand, with special requirements on data exchange speed and connection reliability. Improved security measures in mobile banking put also more pressure on the mobile networks' response time and trust in delivering data. There are several technologies to support the increasing presence of different objects connected to Internet, such as LoRaWAN, ZigBee, Bluetooth, and others, but when discussing about applications developed on smart devices (smart phones, tablets, smartwatches etc.), most of these employ GSM networks and associated services to exchange data. Therefore, increased demand for robustness, rapid recovery and reliability are required for the mobile communication networks and their associated services. In its document [1], ITU-T Focus Group on Disaster Relief Systems, Network Resilience and Recovery describes ways to improve network resilience and recovery time against disasters.

[1] Ph.D. Stud., Orange Services Romania S.A., Romania, e-mail: viviana.minea@orange.ro
[2] Assoc. prof., Dept. of Telematics and Electronics for Transports, Transports Faculty, University POLITEHNICA of Bucharest, Romania, e-mail: marius.minea@upb.ro
[3] Prof., Dept. of Engineering and Management for Transports, University POLITEHNICA of Bucharest, Romania; augustin.semenescu@upb.ro

Based on a layered structure, communication networks can improve robustness against disasters from two aspects: service/application and infrastructure. The document states that "Network resilience is the robustness of the network infrastructure and should ensure the continuity of telecommunication services against any damage caused by the disaster". However, we consider that in the above-mentioned conditions of development, resilience should be an attribute of networks even in normal operating conditions, to cope with abnormal behavior of client applications, variable demands, variable data flowing and so on. As a definition, resilience of a communication network means the ability to provide and maintain an acceptable level of service in conditions of faults and challenges to normal operation. In this paper only the efficiency of application deployment and monitoring activities are analyzed. In the final part of the paper, a solution to improve operational activities is proposed, in order to maintain the Network Resiliency and Recovery (NRR) at high levels. Presently, a rich scientific literature on this subject is being developed. The authors of [2] present definition and storage of fault isolation specific rules, a layered stack of rules, and a network configuration database, for reconfiguring alternate routes with level of service monitoring, followed by decision regarding the healing methods. These are described in a comparatively manner, with both centralized and decentralized approaches for the fault management. The fault management is a complex task that is considered in [3] for the mobile communication networks. Here, the authors present opinions regarding the difficulty of introducing artificial intelligence in 5G and next generation networks, with the purpose of surmounting significant technical barriers in terms of robustness, performance, and complexity. A study on the artificial intelligence algorithms used in smart machine tools' fault management is presented in [4], referring to classifying and presenting conclusions of over 160 scientific works in this domain. The authors of [5] present a proactive fault detection process and suggest that it should be possible to employ adaptive statistical procedures to detect network faults without the necessity of having a priori models of specific faults. Heterogeneous network's fault management is augmented with a new service in [6] employing intelligent solutions. A.J. Garcia & others [7] propose a solution based on data mining for alarm prioritization in fault management of complex cellular networks, for increasing network resilience. M. Awad is proposing some models based on intelligent fault management, applied to mobile radio access networks [8]. Also, for the domain of smart wireless sensors networks, R. Sokollu and others are describing a novel model for fault management [9], and in [10], [11] the authors discuss about fault management in software-defined networking and fault tolerance. The subject of application management influence on network resilience, however, is not much approached in the literature, while still representing, in our opinion, a very typical case of network level of service decrease, in case of malfunctioning. Also, it is a frequent case of prolonged recovery time, therefore also reducing network resiliency. In this paper, a case study is analyzed for a typical communications network with many applications that have to be both manually and automatically monitored and recovered in case of a failure.

The rest of the paper is organized as following: Section 2 provides a brief description of the software instruments used for application monitoring and management. In Section 3, a practical comparison between containers and virtual machines efficiency is analyzed in a case study considered for a virtual communications network. Section 4 proposes a solution to be developed for improving the quality-of-service management, and final conclusions in Section 5.

2. Instruments for application monitoring and ensuring quality of services

Presently, mobile communication networks encompass many hardware and associated software applications and services. The management of these functional components is a complex task, therefore specific artificial intelligence-driven over-imposed services are installed and assist human operators in delivering good level of service in monitoring applications and hardware devices. For obtaining a high level of NRR, some objectives must be taken into consideration:

- Ensuring essential hardware redundancy – "graceful degradation".
- Monitoring and ensuring congestion control – preventive maintenance.
- Procedures for rapid recovery of failed, or slow responding applications involving reduced level of service – operational reliability.
- Adequate procedures for deployment of new features, with minimum impact over other applications and services.
- AI-driven assistance on applications and hardware monitoring, able of producing correlations and recommendations for rapid recovery after major failures – fast, assisted post-event response and recovery processes.

However, today large mobile communication networks with over one million subscribers employ different over-imposed AI applications to support operators in ensuring a reliable quality of service and monitoring applications and services for good operation. Such an instrument for monitoring mixed hardware / service availability is the Application Performance Index (or Apdex), which is expressed by:

$$API_A = \frac{n_s + 0.5 \cdot n_t + 0 \cdot n_u}{N} \tag{1}$$

$$N = n_s + n_t + n_u \tag{2}$$

In the equations (1) and (2), API_A represents the Application Performance Index of the respective application. This index is ranging from 0 to 1. The other terms are as following: n_s the number of satisfactory service level counts, n_t the number of tolerable service level counts, and n_u the number of unsatisfactory service level counts. N is the total number of samples.

Considering how fast technologies develop and platforms upgrade to newer versions, the compatibility will always be a vital problem that needs to be taken into consideration.

Security is also a big concern, mainly for those applications that operate with delicate personal user's information. In today's immersive technologies world, an important place when it comes to running applications is represented by Docker. Docker primarily uses the so-called container technique, which is presently gaining increasing attention and has become the preferred alternative to traditional virtual machines.

Software as a service (SaaS) is a software delivery model where both the software and the associated data are centrally hosted in a cloud. According to a study conducted by North Bridge Venture Partners, "45% of businesses say they already, or plan to run their company from the cloud – showing how integral the cloud is to business". Docker is a tool that makes use of isolated resources that allow applications to be packaged in containers with all the dependencies installed and ran wherever is needed, completely independent of the rest of the host resources. This procedure frees local processing resources and delivers more flexibility. Docker containers are implemented by using virtualization, but the biggest differences between this solution and the alternative of deploying applications on separate virtual machines are the efficiency of resource sharing, flexibility, convenience, lightweight operations, and maintenance. Instead of installing the operating system as well as all the necessary software in a virtual machine, docker images can be easily built with a Dockerfile, which specifies the initial tasks when the docker image starts to run. A container is very similar to an application, which runs as an isolated process on top of the operating system (OS) in its own address space. More than a normal process, a container not only includes the application executable (for example the jar file), but also packs together all the necessary software that the application needs to run with such as the libraries and the other dependencies. The resources allocated to each container can be adjusted dynamically, and the container cannot use more resources than being specified in a control-group mechanism.

Performance comparison between containers and virtual machines has attracted research [12-16]. Rapid recovery of failed applications can be much easier performed and would require a reduced level of service when using a dynamic environment based on Docker orchestration.

3. Comparation between Containers and Virtual Machines Efficiency – a Case Study

3.1 Test conditions

A comparative study of containers efficiency over virtual machines has been performed for a virtual communications network environment.

Two critical services were considered, namely A and B, with a high target availability, that need to be restored as soon as possible. Service A was deployed via Linux virtual machines in a classical manner (by running a process in the background of the servers – Fig. 1), while Service B is a multi-container Docker application (as shown in Fig. 2).

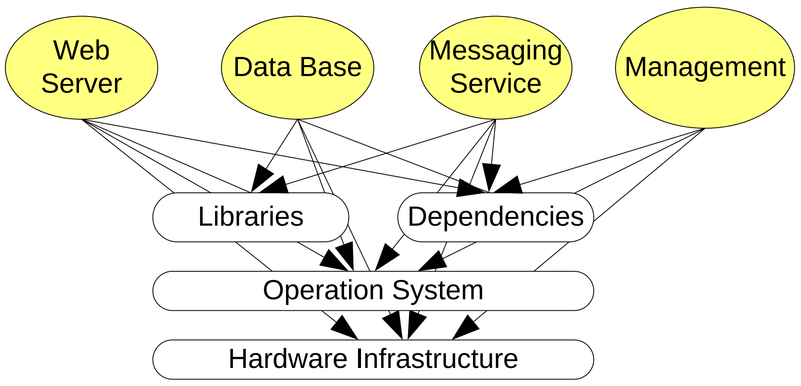

Fig. 1 Deployment of a Jawa application in a classical architecture

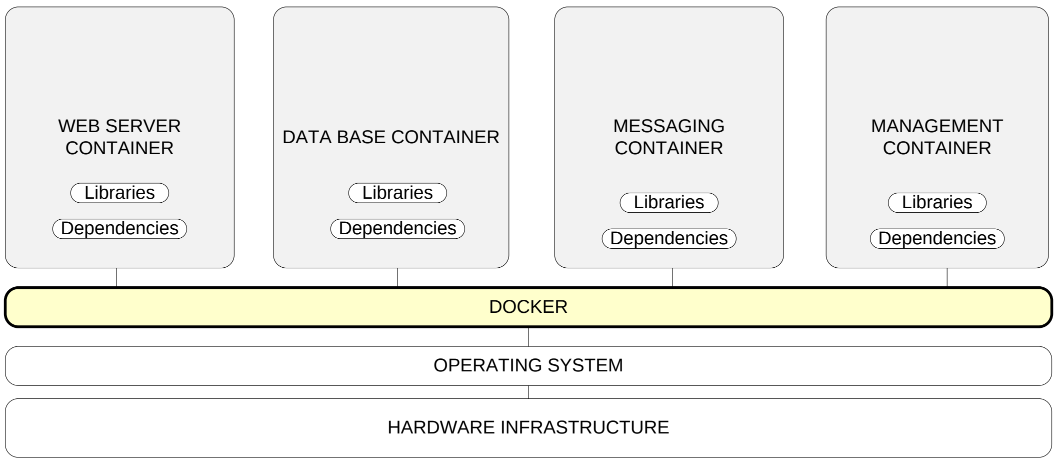

Fig. 2 Deployment of an application employing Docker architecture.

The need of securing a disaster-recovery environment for service A involves the need of having an independent machine where the administrator shall install all the services' prerequisites and then run the process. For service B, the administrator is capable to program the application for running multiple workloads on the same OS, these being independent instances of the same service that do not interfere with each other. If an auto-restart mechanism for these two services is to be implemented, in case of A the administrator has to transform the application in a Linux service using systemd[4] software. Ubuntu has a built-in mechanism to create custom services, enabling them to get started at system boot time or when their process is terminated, and start/stop them as a service. This means a bash script[5] must be written to control the application's state, and this script then needs to be configured as a service using Ubuntu's system and service manager (systemd). All these actions must to be performed manually by the service administrator, because systemd is configured exclusively via plain-text files.

In the case of service B, the Docker orchestration tool is always aware of the state of the application-containers, due to the embedded health monitors. To perform an automatic restart of a Docker container at a crash or system reboot, the only step required is to use a special Docker command. Docker provides restart policies to control whether containers start automatically when they exit, or when Docker restarts. It is a very common use case to add the restart policy on an existing container, involving a very reduced level of service from the administrator's part.

3.2 Test results

Let us consider a generic Java test application called app.jar. This application is then started and checked over performance in a container environment versus a virtual machine environment. Both the experimental virtual machine environment and the container environment consist of Linux virtual servers with 16GB RAM memory and 6 CPU cores of x86_64 architecture, model name Intel(R) Xeon(R) 2.20GHz. The traditional environment runs Java openjdk1.8 and the container environment run Docker version 17.03. The containers are configured to use 100% from the hardware resources.

Aspects of interest for this case study include the following:

- How much resources (in terms of memory usage and CPU power) imply the different implementations.

[4] "Systemd" is a software suite that provides an array of system components for Linux operating systems.

[5] A Bash script is a plain text file which contains a series of commands.

- How convenient the execution environment can be set up, different workloads running in each setup.
- How well each environment can scale. A shorter setup time not only indicates an easier job for the system administrators, but also brings shorter latency, thus better experience to the users.

Runtime system level metrics, such as resources utilization, were collected and compared, along with the average CPU utilization on both environments. This was calculated by averaging values from 4 VMs on which the Java app.jar process has been run and averaging the values from 4VMs on which the Java app.jar was run from containers. Memory usage was sampled every 2 hours during a day and CPU usage every 30 minutes over the same day. The measured values represent the total CPU processing power consumed, composed of user-level CPU, system-level CPU and CPU wait-time. Table 1 below represent a sample of the recorded data during testing period, with memory usage for the Java running server and Docker.

Table 1

Utilization of memory in the studied cases (sample data)

Time	00:00	02:00	04:00	06:00	08:00	10:00	12:00
(Idle RAM usage, Java running server: 850 MB, 6% CPU.) Used RAM Memory [GB]:	11.14	11.11	11.12	11.14	11.13	11.13	12.91
(Idle RAM usage, Docker running: 1.5 GB, 2.6% CPU) Used RAM Memory [GB]:	10.9	10.8	11	11	11	11.2	10.2

Instant fluctuations of values according to time moments that were noticed and recorded are usually a normal behavior, being happening because the processor usage increases or decreases according to momentary job requirements. However, it presents interest to comparatively determine the average usage of the processor in both situations (VM with Java vs. VM with Docker). In normal operation, if there is no contention, any container freely uses all the host's CPU time. For example, if a second container is idle, a first container can make use of all CPU cycles, even though it was configured with fewer shares.

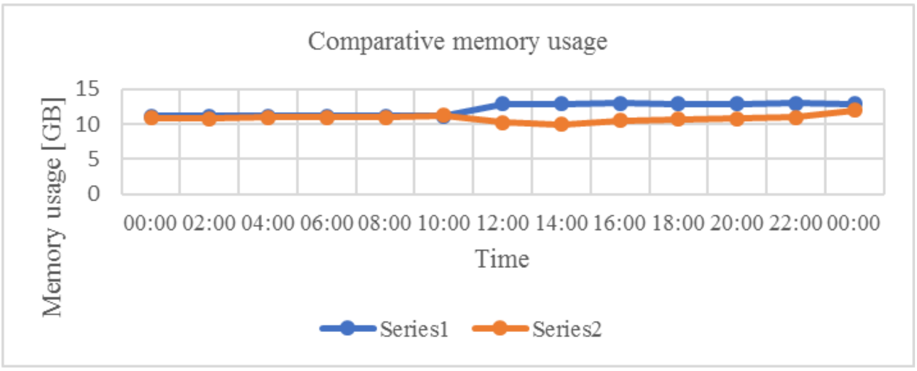

Fig. 3 Memory consumption for the two cases analysed

From Table 1 and Fig. 3, it can be concluded that in the two studied cases the memory usage is better for Docker use case.

Table 2

Utilization of CPU processing power in the studied cases (sample data for Java)

Cluster with 4 servers running Java							
Timestamp	0:00	0:30	1:00	1:30	2:00	2:30	3:00
Server1 in test	20.5	15	40	30	27	27	11.6
Server2 in test	8.2	14.6	14.6	24	25	7.4	9.46
Server3 in test	42	42	42	50	51	51.8	41
Server4 in test	12	14.15	14	40	46	24	24
Average values [%] CPU usage (With stopped app: 7% CPU)	20.675	21.4375	27.65	36	37.25	27.55	21.515

Table 3

Utilization of CPU processing power in the studied cases (sample data for Docker)

Cluster of 4 servers running under Docker							
Timestamp	0:00	0:30	1:00	1:30	2:00	2:30	3:00
Server1 in test	16.6	13.4	14.85	13.35	12.91	15.27	13.7
Server2 in test	5.4	5.46	4.86	5.53	4.86	6.45	4.87
Server3 in test	5.7	5.84	6.9	6.38	6.3	6.43	5.27
Server4 in test	11.5	11.62	11.44	10.95	10.87	10.4	10.8
Average values [%] CPU usage (With stopped app: 2.6% CPU)	6.925	**6.175**	**6.6525**	**6.315**	**6.0175**	**7.0375**	**5.96**

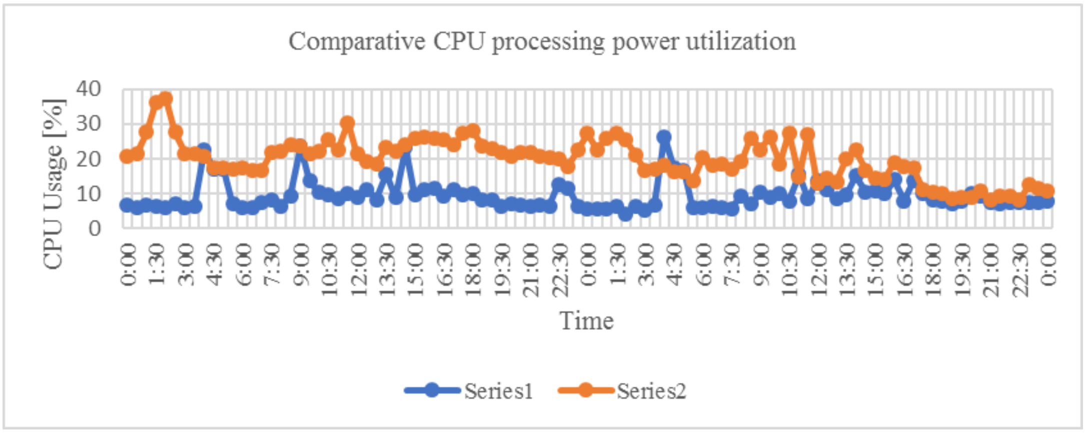

Fig. 4 CPU processing power consumption for the two cases analysed (with Java running server-Series2 and with Docker containers-Series1)

Looking on the test results, it can be noticed that performance of Docker containers and VMs in idle state, considering memory usage gives a bonus to VMs, while considering the CPU usage, Docker containers perform much better (Fig. 4).

When running in container mode, Docker containers perform better mostly when CPU experiences intensive usage. The results also show that compared with virtual machines, containers provide a more easy-to-deploy and scalable environment and a more efficient way of resource sharing. This is a serious remarque, as in maintenance mode that mode of operation eases the fast recovery of failed applications. Still, based on several analysis on communications networks' recovery and maintenance procedures, we can consider that further developing AI applications with machine learning features, able to improve themselves in terms of performance, might contribute significantly to a better network resilience. In this sense, a machine learning engine could be the bond between the human operators and the equipment side, taking advantage of new solutions discovered for solving more complex failures and adding them to a local failure database.

4. Solutions for Improving the Quality-of-Service Management

The development of large communications networks involves many hardware modules and applications deployment. These include permanent maintenance activities, with an increasing pressure on the technical personnel. With the introduction of AI in the maintenance process, the efficiency of maintenance operations can be significantly improved. Davis AI[6] is already functional in many large communications networks. However, new deployments of applications, running software updates, moving databases on different hosts, or performing any other maintenance/development activities in such complex functional systems might trigger unexpected chain malfunctions. From this point of view, we consider that a structure as the one presented in Fig. 5 might help the preventive maintenance activity. The modules drawn in grey are responsible with the intelligent part of the maintenance process. The MLS (Machine Learning Superstructure) shall be able to make use of both positive results from Davis AI operations and human-performed ones, building a new database with complex information on rapid recovery of failed applications. Presently, this solution is under development in separate research and will be subject of a future article.

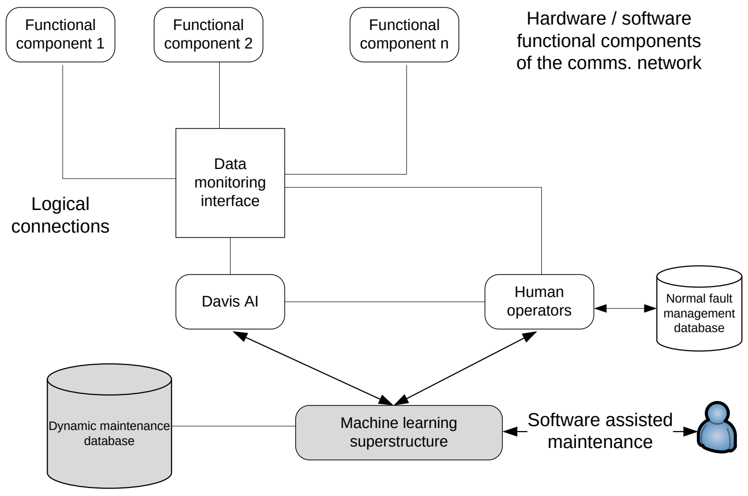
Functional component 1
Functional component 2
Functional component n
Hardware / software functional components of the comms. network
Data monitoring interface
Logical connections
Davis AI
Human operators
Normal fault management database
Dynamic maintenance database
Machine learning superstructure
Software assisted maintenance

Fig. 5 Proposed over-imposed AI, machine learning-driven superstructure for preventive maintenance activities

5. Conclusions

This purpose of this paper was to investigate the benefits of employing containers for applications deployment and maintenance instead of virtual machines, in terms of CPU usage and memory efficiency. The study has been performed for 4 instances of one application installed on servers in the two different cases and run for 48 h continuously, with sampling each 30 minutes for CPU processing power usage and every 2 hours for memory usage. From the

[6] Davis AI is a complex artificial intelligence engine used for analysing millions of connections between software components and associated data, designed to monitor level of service and perform alarms to maintenance operators

results the conclusions are that containers offer much better performances especially for intense tasks which is generally recommended in highly responsive communications networks for a better level of service. Necessary operations flow for an application manager have also been analyzed. There has been noticed a decrease in necessary administrator operations, which frees time for other tasks to human operators, also in the benefit of the quality of service. This study is the first part of a more extensive experiment of developing an over-imposed structure for fault management, as presented in the last part of the work, in order to make more efficient the applications administration.

R E F E R E N C E S

[1]. *ITU-T*, Requirements for Network Resilience and Recovery. FG-DR&NRR - Focus Group Technical Report, Version 1.0 (05/2014)

[2]. *H. Shimazaki* and *N. Takahashi*, NEOPILOT: an integrated ISDN fault management system, [Proceedings] GLOBECOM '90: IEEE Global Telecommunications Conference and Exhibition, San Diego, CA, USA, 1990, pp. 1503-1507 vol.3, doi: 10.1109/GLOCOM.1990.116742.

[3] *R. Shafin, L. Liu, V. Chandrasekhar, H. Chen, J. Reed* and *J. C. Zhang*, "Artificial Intelligence-Enabled Cellular Networks: A Critical Path to Beyond-5G and 6G," in IEEE Wireless Communications, vol. 27, no. 2, pp. 212-217, April 2020, doi: 10.1109/MWC.001.1900323.

[4] *Chang, C.-W.; Lee, H.-W.; Liu, C.-H.* A Review of Artificial Intelligence Algorithms Used for Smart Machine Tools. Inventions 2018, 3, 41. https://doi.org/10.3390/inventions3030041

[5] *Cyntia S. Hood, C. Ji*. Proactive Network-Fault Detection. IEEE Transactions on Reliability, Vol. 46, No. 3, 1997 September

[6] *S. Jiang, D. Siboni, A.A. Rhissa, G. Beuchot*. An Intelligent and Integrated System of Network Fault Management: Artificial Intelligence Technologies and Hybrid Architectures. IEEE Catalogue No. 95TH80610-7803-2579-6/95, 1995

[7] *A. J. Garcia, M. Toril, P. Oliver, S. Luna-Ramirez, M. Ortiz*. Automatic Alarm Prioritization by Data Mining for Fault Management in Cellular Networks. Elsevier Science Direct, Expert Systems with Applications, 158 (2020) 113526, 2020

[8] *M. Awad, H. Hamdoun.* A Framework for Modelling Mobile Radio Access Networks for Intelligent Fault Management. 2016 Conference of Basic Sciences and Engineering Studies (SGCAC), 978-1-5090-1812-3/16, 2016

[9] *R. Sokollu, O. Karaca.* Fault Management for Smart Wireless Sensor Networks. 2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing, 978-0-7695-4843-2/12, 2012

[10] *Y. Yu, X. Li, X. Leng, L. Song, K. Bu, Y. Chen, J. Yang, L. Zhang, K. Cheng, X. Xao*. Fault Management in Software-Defined Networking: A Survey. IEEE Communications Surveys & Tutorials, Vol. 21, No. 1, First Qarter 2019

[11] *Chen, Jue & Chen, Jinbang & Xu, Fei & Yin, Min & Zhang, Wei*. When Software Defined Networks Meet Fault Tolerance: A Survey. 351-368. 10.1007/978-3-319-27137-8_27, 2015.

[12] https://developer.ibm.com/technologies/containers/articles/containerization-docker-case-study/

[13] *A. Babu, M. Hareesh, J. P. Martin, S. Cherian, and Y. Sastri*. System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and

xenserver. In Advances in Computing and Communications (ICACC), 2014 Fourth International Conference on, pages 247–250. IEEE, 2014.

[14] *J. Bhimani, Z. Yang, M. Leeser, and N. Mi.* Accelerating big data applications using lightweight virtualization framework on enterprise cloud. In High Performance Extreme Computing Conference (HPEC), 2017 IEEE, pages 1–7. IEEE, 2017.

[15] *P. R. Desai.* A survey of performance comparison between virtual machines and containers. ijcseonline. org, 2016.

[16] *S. Soltesz, H. P¨otzl, M. E. Fiuczynski, A. Bavier, and L. Peterson*. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In ACM SIGOPS Operating Systems Review, volume 41, pages 275–287. ACM, 2007.

[17] *W. Felter, A. Ferreira, R. Rajamony, and J. Rubio*. An updated performance comparison of virtual machines and linux containers. In Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On, pages 171–172. IEEE, 2015.

[18] *Qi Zhang, Ling Liu, Calton Pu, Qiwei Dou, Liren Wu, and Wei Zhou*. A Comparative Study of Containers and Virtual Machines in Big Data Environment IBM Thomas J. Watson Research, New York, USA; College of Computing, Georgia Institute of Technology, Georgia, USA; Department of Computer Science, Yunnan University, Yunnan, China

[19] https://www.business2community.com/strategy/saas-market-trends-for-2019-and-how-to-align-your-growth-strategy-02128313