*Proc. of the 2nd International Conference on Electrical, Communication and Computer Engineering (ICECCE)*
*12-13 June 2020, Istanbul, Turkey*

# Management of Virtualization Technologies with Complex Event Processing

Elif Cansu Yildiz,
*Link Bilgisayar R&D Center*
*Istanbul, Turkey*
*eyildiz@linkbilgisayar.com.tr*

Doruk Eren Aktas
*Computer Engineering Dept., YTU*
*Istanbul, Turkey*
*doruk.eren.aktas.std@yildiz.edu.tr*

Engin Unal
*Link Bilgisayar R&D Center*
*Istanbul, Turkey*
*eunal@linkbilgisayar.com.tr*

Hakan Tuzun
*Link Bilgisayar R&D Center*
*Istanbul, Turkey*
*htuzun@linkbilgisayar.com.tr*

Mehmet S. Aktas
*Computer Engineering Dept., YTU*
*Istanbul, Turkey*
*aktas@yildiz.edu.tr*

***Abstract*—With the increase of data flow obtained from application virtualization platforms, requirements such as detecting patterns within such data and producing resource management actions have become an emerging need. In this study, a software architecture for a complex event processing module has been proposed and implemented for the analysis of data obtained from application virtualization environments. The prototype of the proposed module is discussed in detail. The performance of the implemented module was evaluated and the results were observed to be promising.**

***Keywords—Application virtualization environments, container, container management system.***

## I. Introduction

Nowadays, large amounts of data are available from various sources and many applications require actions to be produced by processing and analyzing the data. Different applications aim to provide monitoring and management capabilities/actions by processing and extracting the data obtained from different sources such as web sites, mobile applications and sensor data.

Today, an application virtualization platform is a widely used cloud computing technology. By deploying and running a software within an application virtualization platform, we can use the software as a service. With the widely usage of cloud computing, studies have been performed on the performance of application virtualization environments. A widely used tool for an application virtualization environment is Docker [1]. Today, most of the services hosted on application virtualization environment use Docker technology.

There is a need for real-time pattern detection within the streaming data obtained from different virtualization platforms such as Docker containers. Here, critical situations should be determined and actions should be created to address such situations. For instance, it is necessary to create actions to increase or decrease the resource limits in accordance with the use of system resources on a given application virtualization platform.

Currently, data analysis can be done by using two different traditional data processing techniques: stream-data processing and batch-data processing. These techniques are implemented by different software frameworks using various programming paradigms such as map-reduce programming. In this study, we are interested in stream-data processing and analyzing data within large amount of continuous flowing data from varying sources such as application server logs, application logs or activity logs in applications.

There are many situations where millions of data items (i.e. events), coming from different sources, need to be analyzed as quickly as possible. With the complex event processing method, it is possible to detect desired patterns within the flowing data. Applications running in application virtualization environments generate continuous data flow (i.e. performance data collected from containers). The performance data obtained from these sources can be analyzed in a timely manner and actions can be generated based on the predefined rules. This way, hardware resources (cpu, memory, bandwith etc.) can be managed more effective and less costly.

Within the scope of this research, a complex event processing based module has been designed and developed for resource management in application virtualization environments. The developed module uses state data obtained from application virtualization platforms and actions are generated based on predefined rules. To implement the resource management tool, Docker technology and Apache Flink [2] are used as an

978-1-7281-7116-6/20/$31.00 ©2020 IEEE

application virtualization platform and complex event processing tool respectively.

In this manuscript, Section 2 gives an overview of the related work, followed by a discussion of the proposed methodology in Section 3. Section 4 discusses the details of the prototype software, while Section 5 provides an evaluation of the prototype. Section 6 concludes the paper with a summary of the research.

## II. Related Work

The complex event processing technique deals with detection of patterns among multiple streaming events and consists of three steps: input, processing and output.

The input step consists of the collection of the flowing data and data traffic defined as “stream of event". As input of the system, data coming from different sources such as the sensors of the devices can be used. The input step also can also involve collection of data resulting from the intermediate processing of the data.

The processing step is the step in which these events are interpreted according to predefined patterns. Patterns within the flow of events can be detected by processing the events that occur in a certain order, number and frequency. Semantic context awareness can be created by processing the occurrence of events at the same time, the same user and the same location. By monitoring the events that have occurred or not in a given period of time, established temporary awareness can be built.

The output step is the stage where the relevant systems and users are informed by creating an "action" according to the predefined rules.

Apache Flink is a data streaming engine that enables communication over streams, fault tolerance, and data distribution (https://flink.apache.org/). Figure-1 shows the software design of the Flink. Flink's core runtime engine appears to be the flowing data engine, and both the DataSet and the DataStream API are started by this engine. With the Complex Event Processing library in the DataStream API, the flowing data is processed according to the predefined patterns.

Docker is the container technology that enables applications to be packaged, migrated and run in one place with the requirements of the applications (https://www.docker.com/). This technology provides insulation between the conveying machine and other containers. This feature is very similar to classic virtual machine virtualization.

Docker containers are managed by the Docker Engine. Docker engine can be operated by using a RESTful service and status information can be obtained. Using the RESTful interface, limits of resources such as memory, processor, storage, Ethernet can be managed on the containers, container creation, deletion, move and update operations can be performed.

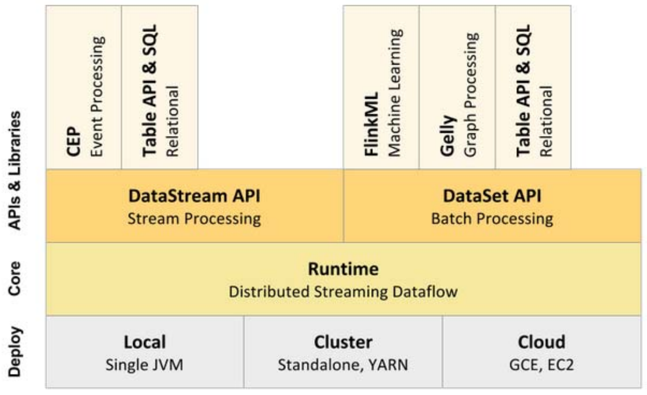


*Figure 1- Flink Component Stack*

Literature Review: The authors of the study [3] utilized complex event processing in their research and stated that multiple and different situations can be analyzed efficiently and actions can be generated such as sending e-mails or sms messages. Here, a complex structure is defined by simple rules and the events in these services are monitored more effectively. Cloud computing enables a system to scale up the necessary system resources for the processes to be used and to make them available when necessary. Dynamically reserving, migrating and freeing system resources is critical as requirements of the systems change. If these requirements cannot be met in a timely manner, this may cause related services to crash. Here, complex event processing is used to analyze the data in order to realize resource management and obtain errors in the system [4]. The authors of the study [5] uses both Apache Flink as complex event processing engine and Apache Kafka as communication bus, in making decisions on the events occurring. The authors of the study [6], provides a comparison of real-time data processing technologies such as Spark, Samza, Storm and Flink. Their study indicates that Apache Flink is superior to others in real-time operation and provides access to historical data, and API support. Our study uses complex event processing feature to detect patterns within the real-time performance data collected from containers used in cloud computing. Docker was used as the container based virtualization technology. We also use the integration of both Apache Flink and Apache Kafka to process events coming from Docker containers.

The complex event processing technique detects patterns in different domains such as social media [7-10], internet of things [11-12], cloud computing [13], e-commerce [14] and real time streaming based grid applications [15]. Our study focuses on detecting predefined patterns within the data collected from container-based virtualization environments.
To keep track the activities of processes and services, provenance based systems has been emerged in recent years [16-20]. In this study, our main focus is the detection of patterns in the events occurring in application virtualization platforms.

There exists many studies that focus on designing and implementing service oriented architecture based systems with a focus on high performance and scalability [21-29]. In this study, our main focus is the use of complex event processing for detecting predefined patterns on the performance data collected from application virtualization environments.

## III. Proposed Complex Event Processing Module

The architecture of the proposed module is illustrated in Figure 2. As shown in Figure 2, performance data (network usage, processor usage, memory usage, network limit, memory limit) is continuously collected from Docker Containers by using the publish/subscribe architecture.

The proposed module takes advantage of the Apache Kafka technology as a publish/subscribe system and uses Apache Flink technology as a complex event processing library. The proposed module generates warnings when it finds a pattern while analyzing real-time data according to predefined rules. The data structure, methods and usage scenarios used by this module are given below, in respective order.

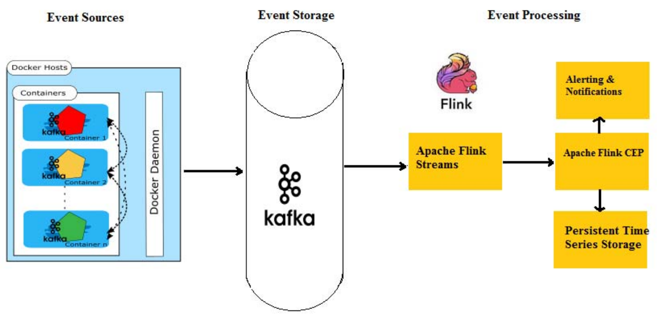


*Figure 2 - Complex Event Processing Software Architecture*

**Data Structures:** Each status data of the containers in Docker technology is obtained in real time via the Docker Engine.

Each incoming data (Table-1) contains the following information: time, network usage, input-output limit, node ID, container ID, input-output usage, network limit, ram usage, ram limit.

*Table 1- Complex Event Processing Module Data Structures*

| Time | Net usage | io lim | Node id | cont | io usg | Con perc | Net lim | Ram lim | Ram usg |
|---|---|---|---|---|---|---|---|---|---|
| 1568 | 314.0 | **0** | pqbw | c08 | 160 | **0.02** | 314 | **818** | **622** |
| 1741 | 255.0 | **0** | hfrt | g7k | 185 | **0.25** | 320 | **818** | **622** |

We use publish-subscribe based messaging bus (Apache Kafka) for transferring the data. When the module detects a pattern and produces a warning, it publishes the warning message to a Kafka topic and transfers it through the Kafka message transmission channel.

**Programming Interface:** We provide a programming interface for the proposed complex event processing module that can manage the application virtualization environments. The proposed module is based on a distributed system architecture. It provides the appropriate actions according to the predefined rules.

**Usage Scenarios:** With the capabilities of the Complex Event Processing module, it is possible to analyze and detect patterns within the streaming data.

For an example; let us assume that there exists a rule, which indicates 80% for the memory limit and 90% for the CPU usage limit. Let us also assume that the action for this rule is set to increase resources for both memory and CPU. Whenever, a pattern, conforming to this rule, is formed, the action corresponding to this rule outputs the result to a Kafka topic. On receiving the message from Kafka, the proposed model increases the memory and CPU resources on the container via Docker Engine.

For another example; let us assume that there is a rule, which requires the initial allocated memory allocation limit to be reduced when the average memory usage is less than 10% of the memory allocation limit. If the container-status matches the pattern indicated in this rule, the system generates an output and publishes it to a Kafka topic. On receiving the message from the Kafka topic, the system decreases the memory allocation limit via the Docker Engine.

## IV. Prototype Implementation

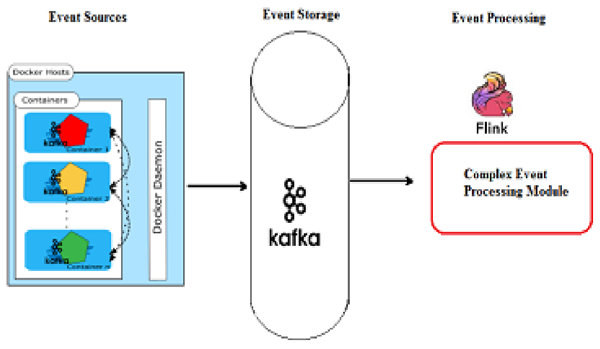


*Figure 3- Prototype Application Architecture*

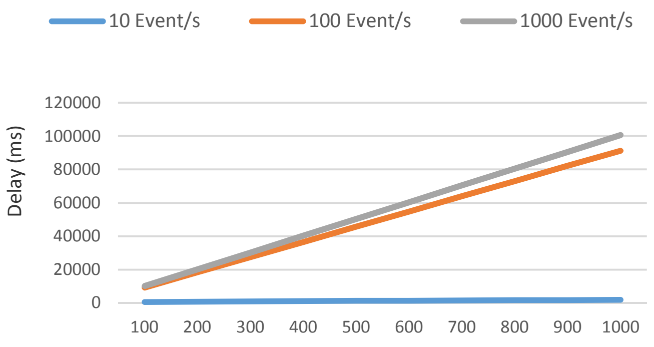
Figure-3 illustrates the prototype application of the proposed architecture. Apache Kafka is used as a message delivery tool. In this application, real-time analysis of container data is aimed to follow up complex situations and provide warnings suitable for complex situations.

## V. Evaluation of The Prototype Implementation

We perform tests on the prototype software on a virtual machine equipped with a Linux Operating System (Ubuntu 16.04) with 4 CPUs and 10 GB of memory. Apache Flink version 1.9.0 was used for the tests. The tests seek answers to the following questions: What is the behavior of the system against the increased message load? How do the reaction times of the sub-modules of the system change in response to the increased message load?

We conduct a test that evaluates the system performance under increasing system loads. The system is tested under 3 different loads (10 event/sec, 100 event/sec, 1000 event/sec).

We record the delay under different loads while the number of messages sent to the system increases. The results show that system performance increases in direct proportion with increasing message load. It is recommended to increase the number of nodes in the system if the delay time requested from the system exceeds the requested values while the number of messages sent to the system increases.



## VI. Conclusions

In this manuscript, a complex event processing based tool is designed for application virtualization environments. A prototype application is implemented and tested on source data obtained from container-based virtualization environments. The performance of the prototype software was evaluated under varying message loads and the results were observed to be successful. The results show that the developed module is successful and useful in detecting complex events collected from application virtualization environments.

## Acknowledgement

We thank Tubitak for supporting this research (under project id: 3118115) and Link Computer R&D Center for providing the computational facilities and datasets that made this study possible.

## References

[1] Fink, J. (2014) Docker: a software as a service, operating system-level virtualization framework, *Code4Lib Journal* Vol: 25.

[2] Carbone, P. et al. (2015) Apache flink: Stream and batch processing in a single engine." *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36(4).

[3] Dhaouadi, J. et al. (2018), On the Data Stream Processing Frameworks: A Case Study. *UBMK-18.*

[4] Büyüktanır, T. et al. (2018), Provisioning System for Application Virtualization Environments, *UBMK-18,* Invited Book Chapter in a Book: Big Data Analytics for Sustainable Computing, Editor: H. Anandakumar, IGI Global, Global Publications, Pennsylvania, pp.1-15.

[5] Javed, M. et al. (2017), Characterization of Big Data Stream Processing Pipeline: A Case Study using Flink and Kafka., *BDCAT-2017*, Texas, USA.

[6] Morshed, S. J. et al. (2016), Open Source Initiatives and Frameworks Addressing Distributed Real-Time Data Analytics, *IPDPSW-16*, Chicago, IL, pp. 1481-1484.

[7] Baeth, M.J. et al. (2019). Detecting misinformation in social networks using provenance data, CONCURR COMP-PRACT E, 31(3).

[8] Baeth M. J. et al. (2018) An approach to custom privacy policy violation detection problems using big social provenance data, CONCURR COMP-PRACT E, 30(21).

[9] Baeth, M.J. et al. (2017). Detecting misinformation in social networks using provenance data, SKG-17.

[10] Baeth, M.J. et al. (2015). On the Detection of Information Pollution and Violation of Copyrights in the Social Web, SOCA-15.

[11] Dundar, B. et al. (2016) A Big Data Processing Framework for Self Healing Internet of Things Applications, SKG-16.

[12] Aktas, M.S. et al. (2019), Provenance aware run-time verification of things for selfhealing Internet of Things applications, CONCURR COMP-PRACT E, DOI: 10.1002/cpe.4263.

[13] Aktaş M.S., (2018) Hybrid cloud computing monitoring software architecture, CONCURR COMP-PRACT E, 30(21).

[14] Aktas M.S. (2019) Detecting Complex Events With Real Time Monitoring Infrastructure On Event-Based Systems, Pamukkale Univ Muh Bilim Derg. 2019; 25(2): 199-207.

[15] Fox, G. et al. (2006). Real Time Streaming Data Grid Applications. Distributed Cooperative Laboratories: Networking, Instrumentation, and Measurements. Editors: Davoli, F. Plazzo, S, Zappatore, S., pp. 253-267.

[16] Riveni, M. et al. (2019). Application of provenance in social computing: A case study, CONCURR COMP-PRACT E, 31(3).

[17] Tas, Y. et al. (2016) An Approach to Standalone Provenance Systems for Big Provenance Data, SKG-16.

[18] Aktas M.S. et al. (2010). High performance hybrid information service architecture, CONCURR COMP-PRACT E, 22(15).

[19] Aktas M.S., et al. (2008) XML metadata services, CONCURR COMP-PRACT E, 20 (7).

[20] Aktas, M.S. et al. (2007). Fault tolerant high-performance Information Services for dynamic collections of Grid and Web services, FUTURE GENER COMP SY, 23(3).

[21] Pierce, M.E. et al. (2008). The QuakeSim project: Web services for managing geophysical data and applications, PURE APPL GEOPHYS, 165(3-4).

[22] Aydin, G. et al. (2005). SERVOGrid complexity computational environments (CCE) integrated performance analysis, GRID-05.

[23] Aktas, M. et al. (2006). iSERVO: Implementing the International Solid Earth Research Virtual Observatory by Integrating Computational Grid and Geographical Information Web Services, PURE APPL GEOPHYS, 163(11-12).

[24] Aydin, G. et al (2008). Building and applying geographical information system Grids, CONCURR COMP-PRACT E, 20 (14).

[25] Oh, S. et al. (2010) Mobile Web Service Architecture Using Context-store, KSII T Internet Info, Volume 4.

[26] Nacar, M.A. et al. (2007) VLab: collaborative Grid services and portals to support computational material science, CONCURR COMP-PRACT E, 19 (12).

[27] Aktas, M.S. et al. (2004). A web based conversational case-based recommender system for ontology aided metadata discovery, GRID-04, pp:69-75.

[28] Aktas, M, (2007) A Federated Approach to Information Management in Grids. INT J WEB SERV RES, 7(1).

[29] Fox, G. et al, (2006) Grids for real time data applications. Parallel Processing and Applied Mathematics, Vol:3911, Book Editors: Wyrzykowski, R and Dongarra, J and Meye, N and Wasniewski, J.