Environment. Technology. Resources. Rezekne, Latvia
Proceedings of the 13th International Scientific and Practical Conference. Volume 2, 62-68

On Integration of Evolving Infrastructure Topology Graphs and Metric Data Streams in Information Technology Infrastructure Management

Jānis Kampars
Riga Technical University
Riga, Latvia
janis.kampars@rtu.lv

Jānis Grabis
Riga Technical University
Riga, Latvia
janis.grabis@rtu.lv

Ralfs Matisons
Riga Technical University
Riga, Latvia
ralfs.matisons@rtu.lv

Artjoms Vindbergs
TET
Riga, Latvia
artjoms.vindbergs@tet.lv

***Abstract* - Modern cloud-based information technology (IT) infrastructure monitoring context and data are gathered from various systems. Typical monitoring systems provide a set of metrics characterizing the performance and health of a variety of infrastructure components. To understand the dependencies and relations among these measurements, the infrastructure topology can be analysed to provide context to the monitoring metrics. However, the metrics and the topology are updated at different time intervals and providing continuous merging and analysis of both data sets is a challenging task which is rarely addressed in the scientific literature. The paper elaborates a method for integration of infrastructure topology graph and monitoring metric data streams. The method is intended for application in the identification of anomalies in IT infrastructure.**

Keywords* - *infrastructure monitoring, infrastructure topology, stream processing, evolving graphs

I. Introduction

Modern information technology infrastructure is highly complex, and it consists of several subsystems such as software defined and physical network, software defined and traditional storage systems, physical servers, hypervisors, container orchestration platforms, and cloud computing platforms. Each of the infrastructure subsystems have a corresponding topology graph of infrastructure components with their corresponding metadata (e.g., allocated RAM for a certain virtual machine or characteristics of drives used in a storage system) and rapidly changing metrics of infrastructure components (e.g., input/output operations per second for a logical or physical drive). A typical large IT infrastructure generates millions of events per day at rates of about 100 events per second [1] and an averaged sized cloud has around 1000 tenants and 100,000 users) [2].

To monitor the entire IT infrastructure as a whole while taking into consideration the interrelationships of certain IT infrastructure components from different subsystems, all topology graphs and component level metrics and their corresponding time series data should be merged and analysed. Such analysis is a computationally and algorithmically complex task since massive amounts of data with different update intervals and data models need to be processed while minimizing the latency. Moreover, upon identifying a certain anomaly, the respective IT infrastructure components such as virtual machines or containers might have already been disposed and therefore removed from the infrastructure topology graph, which is why versioning of the topology graph is required for incident traceability purposes.

The objective of this article is to propose a method for providing infrastructure topology graph versioning and topology aware analysis of infrastructure component metrics.

The paper is structured as follows. Section II presents a method for providing topology aware processing of

Online ISSN 2256-070X
https://doi.org/10.17770/etr2021vol2.6607

infrastructure component time series data and topology graph versioning. Section III proves the applicability of the proposed solution by presenting a practical implementation of the method in Apache Spark, Kafka, Cassandra, and Neo4j. Section IV reviews related research and Section V concludes with final remarks.

II. Method Overview

This section presents a method for integrating evolving IT infrastructure topology graphs and IT infrastructure component related time series data. A high-level overview of the proposed approach is given in Fig. 1.

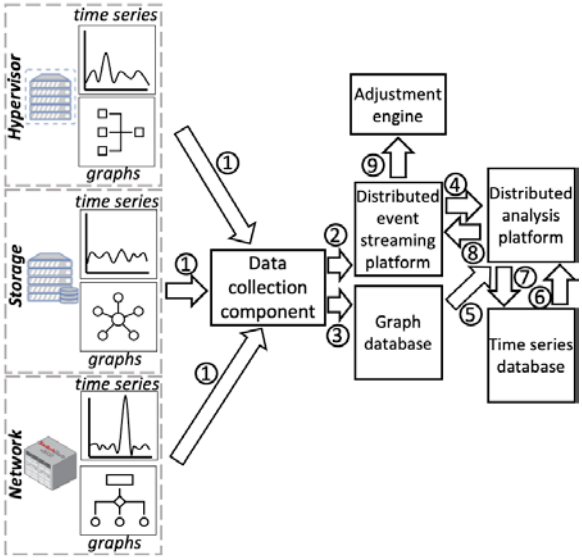

Fig. 1. Overview of the proposed method.

The given example considers three subsystems of IT infrastructure – a hypervisor, storage, and network. Each of them contains an associated topology graph which can be retrieved via API calls specific to the concrete subsystem. Such calls are computationally intensive and put a significant load on the respective subsystem, which is why their frequency needs to be limited depending on the performance characteristics of the specific subsystem. Furthermore, topology graphs can evolve in varying speeds depending on the type of subsystem. For instance, a topology graph corresponding to traditional storage equipment would experience significantly less updates compared to a graph originating from a container orchestration platform such as Kubernetes, where containers are initialized and disposed without any manual intervention. Our experience shows that the full topology graph retrieved from the respective subsystem contains a large number of vertices and edges, which are irrelevant for IT infrastructure monitoring purposes.

There are also time series data originating from the subsystems characterized by high velocity, volume and different schema. It is important that elements whose properties are constantly being measured in the time series data can be linked to certain nodes or edges in the infrastructure topology.

Topology and time series data need to be collected from the respective subsystems, what is done by the data collection component (depicted as connection #1 in Fig. 1). The following approaches can be applied for this purpose:

- Data pull – data collection component constantly queries the respective subsystem to get the time series data or topology graph. This approach is inefficient in terms of performance since subsystems are queried even if no changes have occurred. The advantage of this approach is relying on already existing APIs or log file structure and avoiding customization of infrastructure subsystem management.
- Data push – component of the subsystem or its management layer is customized to send the data to the data collection component upon receival of new data. This allows to distribute the load between the components (such as virtual machines) and achieve higher velocity of the data. It is complex to implement data push in the case of topology graph monitoring, since it would require extending the management layer of the respective subsystem.
- Reverse proxy – this strategy can be applied for continuous versioning of the topology graph without making any changes in the management layer and avoiding putting any additional computational load on the respective subsystem. This can be applied for subsystems where changes in the topology graph are made through a management web service. For instance, virtual machines are built through the web portal of the cloud computing platform, which in turn calls a management REST web service to trigger the creation of a new virtual machine, thus triggering an update in the topology graph. Putting a reverse proxy in front of the REST web service would allow to detect such events and alter the topology graph without directly querying the management layer of the subsystem. This approach can be used for detecting incremental graph updates; however, it would still be necessary to use APIs for establishing the initial state of the topology graph.

The data collection component feeds time series data into the Distributed event streaming platform (depicted as connection #2 in Fig. 1), filters out unnecessary topology graph data, and stores topology graph updates inside a graph database as a combined data centre level topology graph (depicted as connection #3 in Fig. 1). Separate topics are created in the Distributed event streaming platform for each topology component related metric (e.g., a dedicated topic for drive input/output operations per second). Component identifiers form the message key, while measurements are stored in the value section. It is advisable to use event time in the event message.

The distributed analysis platform provides processing of both topology graphs and time series data. It is done based on predefined analysis rules, which regulate:

- graph topology processing for retrieving subgraphs specific to the concrete analysis rule (e.g., all virtual machines and their corresponding hypervisors),
- time series data processing while linking certain infrastructure components based on the topology subgraph (e.g., link misbehaving virtual machine with a connected overloaded logical drive, and its physical drives).

Graph topology processing functionality of each infrastructure analysis rule includes the following:

- Graph query – a query to get the rule-related subgraphs from the graph database (depicted as connection #5 in Fig. 1)
- Graph hash calculation – a function for calculating a hash of a subgraph. This is used for detecting any structural changes in the graph.
- Get a graph ID – a function for calculating a unique identifier for each subgraph. Changed subgraph hash for a particular graph ID indicates structural changes in the specific subgraph.
- Graph serialization – a function of serializing the graph and storing it in the temporal database as a revision of the graph.

Graph topology processing is performed by the Distributed analysis platform and the serialized subgraph revisions, their corresponding hashes, IDs, and timestamps are stored in the Time series database (depicted as connection #7 in Fig. 1).

Time series data analysis functionality of each infrastructure analysis rule is concerned with the following:

- Retrieval of time series data streams from the relevant topics of the Distributed event streaming platform (depicted as connection #4 in Fig. 1).
- Retrieval of the topology subgraph from the Time series database.
- Deserialization of the time series data and topology graph, merging of both data sets according to the logic specified in the rule (e.g., calculation of average disk writes within a single logical disk as the average of all corresponding physical drives).
- Performing windowing operations and aggregations, storing intermediate stream processing results in temporal topics of the Distributed event streaming platform Experiments (depicted as connection #8 in Fig. 1).
- Archiving time series data aggregations in the Time series database for batch processing and later analysis (depicted as connection #7 in Fig. 1).
- Passing information about the detected anomalies as a data stream to a topic in the Distributed event processing platform (depicted as connection #8 in Fig. 1).

A specific stream consumer is created for reacting upon the detected anomalies and it is deployed in the adjustment engine (depicted as connection #9 in Fig. 1).

Hierarchical rules can be created so that an infrastructure rule operating on a higher level of abstraction uses the anomaly feed provided by an infrastructure analysis rule operating on a lower level of abstraction.

III. EXPERIMENTS

To prove the applicability of the proposed approach, a prototype containing a single infrastructure analysis rule is implemented.

Neo4j is used as the graph database to store the joint topology of a storage subsystem (IBM Storwize) and virtualization subsystem (Vmware vCenter). Apache Kafka is used as the Distributed event processing platform. For experiment purposes, the time series data is accumulated in CSV files and a data simulator class is implemented in Python programming language to provide a controlled environment with expectable results and to simulate anomalies according to a predefined experiment plan. Apache Cassandra is used as the Time series database, while Apache Spark serves the purpose of Distributed analysis platform.

The sample infrastructure analysis rule considers identifying anomalies in a physical drive belonging to a common logical drive. This is based on the assumption that the storage subsystem manages to distribute load efficiency between the physical drives forming a logical drive, therefore a notable difference in physical drive performance metrics could be seen as an indication of a faulty drive or an anomaly. The list of monitored metrics, each of which are being streamed to a separate Kafka Topic are given in Table I.

TABLE I. Physical drive metrics

#	**Disk drive metrics**	
	Acronym	***Description***
1.	driveStats.mdsk.pre	Indicates the peak of read external response in milliseconds for each MDisk
2.	driveStats.mdsk.pro	Indicates the peak of read queued response in milliseconds for each drive.
3.	driveStats.mdsk.pwe	Indicates the peak of write external response in milliseconds for each drive.
4.	driveStats.mdsk.pwo	Indicates the peak of write queued response in milliseconds for each drive
5.	driveStats.mdsk.re	Indicates the cumulative read external response in milliseconds for each drive.
6.	driveStats.mdsk.rq	Indicates the cumulative read queued response in milliseconds for each drive

Fig. 2 shows the average `driveStats.mdsk.pre` value within a logical drive, anomaly margins calculated as three standard deviations away from the average value, and individual values for two included physical drives, one of which is an anomaly. It can be observed that drive with identifier `driveStats_124` behaves normally, while the drive with identifier `driveStats_26` is experiencing potentially abnormal behaviour.

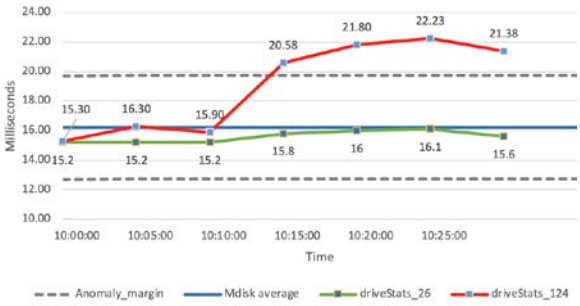

Fig. 2. Anomaly for a physical drive.

The implementation of the graph topology processing logic for the mentioned rule is given bellow.

```
from hashlib import md5
from typing import Type
from ..abstract_rule.AbstractGraphJob import
AbstractGraphJob
from .CassandraModel import DiskAnomalies

class GraphJob(AbstractGraphJob):
 cassandra_model: = None
 ruleName = "disk_anomalies"

 def init__(self, **kwargs):
  self.cassandraModel = DiskAnomalies
  self.graphQuery = """Match (drive:
  storwize_drive)-[:RELATED]
  (mdisk:storwize_mdisk) return
  mdisk.metricTopoId,
  collect(drive.metricTopoId)"""
  AbstractGraphJob.__init__(self, **kwargs)
 def getSubgraphHash(self, subgraph):
  return md5("|". join(subgraph[1]).
  encode('utf-8')).hexdigest()

 def _getSubgraphId(self, subgraph):
  return subgraph[0]

 def _getSerializedGraph(self, subgraph):
  return {'drives': subgraph[1],
  'mdisk': subgraph[0]}
```

The provided name of the rule is used to create Apache Cassandra tables for the current version of the topology subgraphs and their previous revisions. The referenced `DiskAnomalies` class further specifies the data model used for serializing the topology subgraph, while the actual serialization is performed by the function `getSerializedGraph`, which shows that the serialized graph will have two attributes – drives (an array of the physical drives) and mdisk (the logical disk which the physical drives belong to).

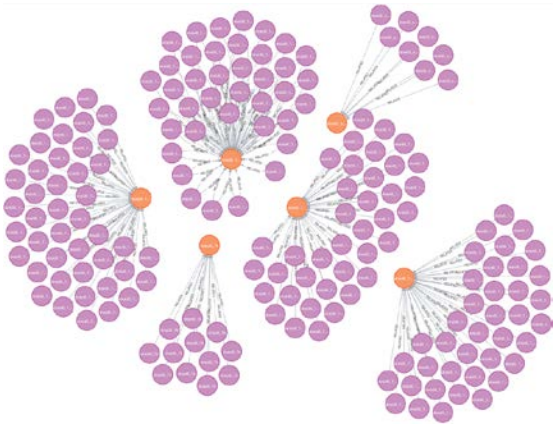

Fig. 3. Matched topology subgraphs for disk anomaly rule.

The graph query specified in `graphQuery` attribute of the class finds all topology nodes which are tagged as “storwize drive” (physical drive) and are connected to a node tagged as “storwize mdisk” (logical disk). Tabular representations of the matched subgraphs are returned, where an identifier stored in the graph node attribute `metricTopoId` is returned for each matched node. Visualization of matching subgraphs is given in Fig. 3, while an excerpt from tabular representations of the matched subgraphs is given in Fig. 4.

mdisk.metricTopoId	collect(drive.metricTopoId)
"ext-virt3-storage_managedDiskStats_80"	["ext-virt3-storage_driveStats_142", "ext-virt3-storage_driveStats_133", "ext-virt3-storage_driveStats_180", "ext-virt3-storage_driveStats_125", "ext-virt3-storage_driveStats_136", "ext-virt3-storage_driveStats_58", "ext-virt3-storage_driveStats_143", "ext-virt3-storage_driveStats_152", "ext-virt3-storage_driveStats_70", "ext-virt3-storage_driveStats_60", "ext-virt3-storage_driveStats_53", "ext-virt3-storage_driveStats_62", "ext-virt3-storage_driveStats_66", "ext-virt3-storage_driveStats_69", "ext-virt3-storage_driveStats_130", "ext-virt3-storage_driveStats_156", "ext-virt3-storage_driveStats_151", "ext-virt3-storage_driveStats_162", "ext-virt3-storage_driveStats_61", "ext-virt3-storage_driveStats_48", "ext-virt3-storage_driveStats_57", "ext-virt3-storage_driveStats_64", "ext-virt3-storage_driveStats_148", "ext-virt3-storage_driveStats_134", "ext-virt3-storage_driveStats_51", "ext-virt3-storage_driveStats_137", "ext-virt3-storage_driveStats_155", "ext-virt3-storage_driveStats_52", "ext-virt3-storage_driveStats_124", "ext-virt3-storage_driveStats_59", "ext-virt3-storage_driveStats_56", "ext-virt3-storage_driveStats_65", "ext-virt3-storage_driveStats_55", "ext-virt3-storage_driveStats_50", "ext-virt3-storage_driveStats_123", "ext-virt3-storage_driveStats_63", "ext-virt3-storage_driveStats_54", "ext-virt3-storage_driveStats_167"]

Fig. 4. Excerpt from tabular representation of topology subgraphs for disk anomaly rule.

The graph ID is equal to the ID of the corresponding mdisk (logical disk), while its hash is calculated as md5 encoded list formed by the identifiers of the included physical drives. The `AbstractGraphJob` class that the topology processing class of the disk anomaly rule extends provides built-in functionality for storing serialized subgraphs in Cassandra tables and creating new revisions upon detected changes in the graph hash. An ER diagram for two tables created to store serialized subgraphs is given in Fig. 5.

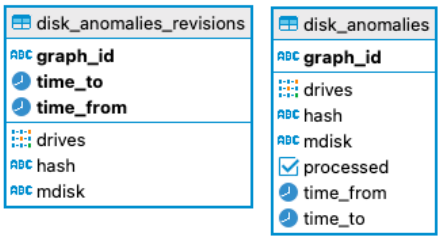

Fig. 5. ER diagram for the two created topology subraph tables.

The `disk_anomalies` table contains the current versions of the existing subgraphs and their corresponding information. Graph processing job tracks the time at which the subgraph appeared first in the `time_from` column, while the `time_to` column contains the last time the specific graph was detected. Following is the pseudocode of the algorithm for versioning of the subgraphs.

```
Update disk_anomalies set processed = False
Select all matched subgraphs from Neo4j
For each returned subgraph do
 If graph_id does not exist in table
 disk_anomalies
  Insert subgraph into disk_anomalies
   Else if graph_id exists in disk_anomalies
    If subgraph hash matches stored hash
     Update time_to to current time, set
     processed = True
    Else if subgraph hash has changed
     Copy stored subgraph to revisions table
     Update the subgraph in disk_anomalies
     table, set processed = True
Select subgraph from disk_anomalies where
unprocessed = True
 For each returned subgraph:
  Copy stored subgraph to revisions table
  Delete stored subgraph from disk_anomalies
```

The time series data analysis is implemented as an Apache Spark job and operates according to the following logic:

- Create a Spark dataframe `anomaly_margins` with drive metric corresponding anomaly margins. It will be evaluation how many standard deviations away is the average value of a drive metric from the average value of the corresponding mdisk (logical disk) metric. If the value is greater than the value specified within the anomaly margin dataframe, it will be considered an anomaly.
- Create a Spark dataframe `drive_mdisks` from Cassandra `disk_anomalies` table.
- Create drive metric Spark data stream `drive_metrics` from Kafka topics that correspond to the physical drive metrics of interest. The data stream contains unprocessed drive metrics as received from the IBM Storwize subsystem.
- Update the `drive_metrics` data stream by joining it with `drive_mdisks` dataframe so that it now contains a corresponding mdisk ID for each drive.
- Create a new Spark data stream `drive_averages` from the `drive_metrics` stream by calculating an average value per drive per metric within a 10 second tumbling window. Save results to a new Kafka topic `da_drive_averages`.
- Create a new Spark data stream `mdisk_averages` from the `drive_metrics` stream by calculating an average value and standard deviation per logical disk, per metric within a 10 second tumbling window. Save results to a new Kafka topic `da_mdisk_averages`.
- Create two new Spark data streams based on the Kafka topics `da_mdisk_averages` and `da_drive_averages` and join them into a new data stream `joined_df` based on the time window, metric name and logical disk, so that for each average metric value of a physical drive there is a corresponding average value of the logical disk and its standard deviation.
- Join the `joined_df` with `anomaly_margins` so that for each metric there is a corresponding anomaly margin available as a new column `stdev_margin`.
- Add a new column `stdev_diff` to the `joined_df` which measures how many logical disk standard deviations away is the physical drive average metric value from logical disk's average value.
- Filter rows from `joined_df` where `stdev_diff` > `stdev_margin`.
- Group by drive and time window, count the number of rows per disk and collect the names of the anomaly metrics inside a new column. Output the results to a new Kafka topic `disk_anomalies`.

During the experiment, the IT infrastructure monitoring process is simulated and anomalies are induced. An extract from a Kafka console consumer connected to the `disk_anomalies` topic is given in Fig. 6. It can be seen that an anomaly is detected for the physical drive `ext-virt3-storage_driveStats_26` and a total of 6 anomalies was observed for the given drive. The names of the corresponding metrics are given in the `anomaly_metrics` array, while `window_start` and `window_end` indicate the start and end of the aggregation window.

```
{"drive_id":"ext-virt3-storage_driveStats_26","mdisk":
"ext-virt3-storage_managedDiskStats_48","graph_id":"ex
t-virt3-storage_managedDiskStats_48"}        {"window_
start":"2021-03-26T18:20:20.000+02:00","window_end":"2
021-03-26T18:20:30.000+02:00","counted_anomalies":6,"a
nomaly_metrics":["driveStats.mdsk.rq","driveStats.mdsk
.pwe","driveStats.mdsk.pre","driveStats.mdsk.re","driv
eStats.mdsk.pwo","driveStats.mdsk.pro"]}
```


Fig. 6. Detected physical drive anomalies.

IV. Related work

Two types of data can be considered when analysing IT infrastructure for the purpose of detecting anomalies and providing predictive maintenance – evolving infrastructure graphs and time series data describing various IT infrastructure components. Existing research papers mostly concentrate on one of the aspects – either topology or time series data analysis. One of the few exceptions is the paper by Kampars et al. [3] concentrating on both data sets. The time series data are referenced as measurable properties, while aggregations are called context elements. The proposed solution lacks the ability to retrieve the topology graph from a data source and it is constructed manually. The system is based on Apache Kafka, Apache Spark and Apache Cassandra. Topology related information is stored in Cassandra and no dedicated graph database is being used.

A. Topology based infrastructure analysis

An example of topology driven anomaly detection can be found in the work by Niwa et al. [4], who present a framework for identifying anomalies in software services of OpenStack cloud computing platform. The framework is implemented in Python and Neo4J is used as the graph database for storing the topology graph.

Topology based root cause analysis of an IT infrastructure failure is also addressed by Schoenfisch et al. [5], who propose a Markov Logic Networks and abductive reasoning based solution. The proposed system was implemented in RoCA, a tool providing a graphical user interface for modelling the infrastructure and conducting the root cause analysis.

Majumdar et al. [2] perform IT infrastructure analysis for security purposes and propose a solution that is able to identify topology inconsistencies that might occur between multiple subsystems of a cloud computing platform. The proposed system gathers data from cloud management systems, cloud infrastructure system, and data centre infrastructure components. The data collection is performed in batch mode.

The security threats caused by cloud platform misconfiguration or insider attacks are addressed by Bleikertz et al. [6]. The authors establish a security system, which proactively analyses the intended cloud infrastructure configuration changes and risks associated with them and then either approves or rejects them. The graph is constantly updated whenever changes in infrastructure configuration occur [7].

A construction of a cloud-based IT infrastructure topology graph is addressed by Mensah et al. [8]. Logs from Cloud Management System and Software Defined Network controller are scanned to detect events that alter the infrastructure topology graph. The proposed system is validated by using OpenStack cloud computing platform.

B. Time series based infrastructure analysis

Harper et al. [1] propose a method for detecting failures of individual infrastructure elements based on the received operational status data and alerts. The work concentrates on detecting cascading infrastructure errors are without any knowledge of the infrastructure topology.

Mijumbi et al. [9] propose a system for analysing communication system alarms. The system is built using Apache Kafka, MongoDB, and python data science tools such as sklearn, pandas, numpy.

Anomaly detection and root cause analysis is also addressed by Lin et al. (2016). The paper proposes a method for virtualized cloud data centres and addresses the scalability challenges by using Apache Spark.

Another clustering-based anomaly detection solution is proposed by Cucinotta et al. [9] The authors perform analysis of system-level metrics, mostly related to resource consumption patterns of virtual machines by using self-organizing maps (SOM) based approach.

V. Conclusion

The paper presents a method for performing IT infrastructure analysis based on both metric time series data and evolving IT infrastructure topology graph. The applicability of the proposed approach is proven by implementing a prototype aimed at identifying physical drive anomalies in IT infrastructure and it is based on Apache Spark, Kafka, Cassandra, Neo4J and Python programming language.

The method allows combing topology data and time series data for comprehensive analysis of anomalies in the complex IT infrastructure. The analysis is performed in real-time and extra computational load on the infrastructure is minimized. The method also uses efficient versioning to track changes in the dynamic topology.

Identification of anomalies depends on predefined rules. These rules are derived by means of data analysis and expert knowledge. In further research, a set of rules will be derived, and the method will be evaluated to determine its computational efficiency and anomalies detection power.

VI. Acknowledgement

This research is funded by European Regional Development Fund Project Nr. 1.1.1.1/19/A/003 “Development of integrated monitoring and predictive maintenance solution for dynamically evolving IT infrastructure” Specific Objective 1.1.1 “Improve research and innovation capacity and the ability of Latvian research institutions to attract external funding, by investing in human capital and infrastructure” 1.1.1.1. measure “Support for applied research” (round No.3)

REFERENCES

[1] R. Harper and P. Tee, "A method for temporal event correlation," *2019 IFIP/IEEE Symp. Integr. Netw. Serv. Manag. IM 2019*, pp. 13–18, 2019.

[2] S. Majumdar *et al.*, *Cloud security auditing*, vol. 76. 2019.

[3] J. Kampars and J. Grabis, "Near Real-Time Big-Data Processing for Data Driven Applications," *Proc. - 2017 Int. Conf. Big Data Innov. Appl. Innov. 2017*, vol. 2018-Janua, pp. 35–42, 2018.

[4] T. Niwa, Y. Kasuya, and T. Kitahara, "Anomaly detection for openstack services with process-related topological analysis," *2017 13th Int. Conf. Netw. Serv. Manag. CNSM 2017*, vol. 2018-Janua, pp. 1–5, 2017.

[5] J. Schoenfisch, C. Meilicke, J. von Stülpnagel, J. Ortmann, and H. Stuckenschmidt, "Root cause analysis in IT infrastructures using ontologies and abduction in Markov Logic Networks," *Inf. Syst.*, vol. 74, pp. 103–116, 2018.

[6] S. Bleikertz, C. Vogel, T. Gross, and S. Mödersheim, "Proactive security analysis of changes in virtualized infrastructures," *ACM Int. Conf. Proceeding Ser.*, vol. 7-11-Decem, pp. 51–60, 2015.

[7] S. Bleikertz, C. Vogel, and T. Groß, "Cloud radar: Near real-time detection of security failures in dynamic virtualized infrastructures," *ACM Int. Conf. Proceeding Ser.*, vol. 2014-Decem, no. December, pp. 26–35, 2014.

[8] P. Mensah, S. Dubus, W. Kanoun, C. Morin, G. Piolle, and E. Totel, "Connectivity graph reconstruction for networking cloud infrastructures," *2017 IEEE 16th Int. Symp. Netw. Comput. Appl. NCA 2017*, vol. 2017-Janua, pp. 1–9, 2017.

[9] R. Mijumbi, A. Asthana, C. Bernal, and M. Castejon, "MAYOR: machine learning and analytics for automated operations and recovery," *Proc. - Int. Conf. Comput. Commun. Networks, ICCCN*, vol. 2019-July, 2019.