Graph-based Diagnosis in Software-Defined Infrastructure

Joseph Wahba, Hazem Soliman, Hadi Bannazadeh and Alberto Leon-Garcia
Department of Electrical & Computer Engineering
University Of Toronto
Toronto, Ontario

***Abstract*—Performing system diagnosis is a critical task in modern datacenters. Investigating individual resource behavior may not be efficient in detecting abnormal behavior in large and complex datacenters. In this paper, we propose a scalable graph based diagnosis framework to detect system anomalies in Software-Defined Infrastructure running in SAVI testbed. We have leveraged Graph Mining and Machine Learning techniques in our approach in order to detect different kinds of anomalies. We have experimentally tested our framework on several use cases: Webserver-Database workload pattern, bandwidth throttling between a pair of VMs, denial-of-service (DoS) attack on a webserver and Spark Job failure. Our framework was able to detect the aforementioned anomalies accurately.**

Index Terms Anomaly Detection, Graph Mining, Machine Learning, Software-Defined Infrastructure, System Diagnosis

I. Introduction

As different cloud platforms continue to grow in scale and complexity, the diagnosis and management task of cloud data centers and platforms becomes a critical challenge. Detecting abnormal behaviors in a data center targets to spot unusual system behaviors such as operator errors, hardware, software failures, different attacks and anomalous communication patterns. Resource based anomaly detection techniques are useful in diagnosing anomalies in individual resources. By leveraging Graph-Mining and Machine Learning techniques, unusual behaviors in data centers could be detected not only based on a per-resource behavior, but using a holistic view of inter-dependency and inter-communication pattern between different resources.

One such cloud platform is the SAVI [1] testbed on which we have implemented our approach. The SAVI project was established to investigate future application platforms designed for rapid applications enablement. SAVI testbed has been developed for controlling and managing converged virtual resources focused on computing and networking. In a SAVI Smart Edge we have compute, network, storage, FPGA, and other resources. OpenStack [2] is used for managing compute, storage, GPU and FPGA resources. OpenFlow [3] controllers are used for controlling network resources such as switches.

Our main contribution is developing a graph-based anomaly detection framework for the SAVI testbed. Our framework leverages the Apache Spark big-data platform for scalability. We have tested our framework on several use cases including Webserver-Database workload pattern, bandwidth throttling between a pair of VMs, denial-of-service (DoS) attack on a webserver and Spark Job failure. Our framework was able to detect the aforementioned anomalies accurately.

The rest of the paper is organized as follows. In Section II, we have surveyed related work. In Section III, we describe our system architecture. In Section IV, we present the design of the diagnosis module. In Section V, we present a proof of concept describing the various use cases of our system. In Section VI, we provide our experiments' results.

II. Related Work

Graph based Anomaly detection has been studied under many different settings using various statistics tools and graph mining algorithms [4]. There are two main categories of approaches for detecting anomalies in graphs: Methods for static graph and approaches for dynamic graph data.

A. Anomaly Detection in Static Graphs

In static graphs, the main task for anomaly detection is to discover anomalous network entities (e.g., nodes, edges) given the entire graph structure. Static graphs are either plain graphs which do not have attributes or attributed graphs where nodes and/or edges have features associated with them. Given a snapshot of a plain or attributed graph, the anomaly detection problem could be defined as finding the nodes and/or edges that are few and different and are significantly different from the patterns observed in the graph historical patterns. In static plain graphs, the only available information is the graph's structure. Therefore, in order to detect anomalies, the structure of the graph is used to find patterns and spot anomalies. There are two main categories of methods in detecting anomalies in static plain graphs: structure-based methods [5] [6] [7] [8] and community-based methods [9] [10] [11] [12]. In static attributed graphs, anomaly detection methods exploit the structure as well as the correlation of attributes of the graph to find patterns and spot anomalies [13] [14] [15]. In community based methods, approaches aim to identify those outlier nodes in a graph that attribute values of which deviate significantly from the other members of the specific communities that they belong to [16] [12] [17].

B. Anomaly Detection in Dynamic Graphs

Dynamic graphs are time-evolving graphs which are composed of sequences of static graphs. Given a sequence of graphs, the anomaly detection problem could be defined as

978-3-901882-85-2 © 2016 Crown

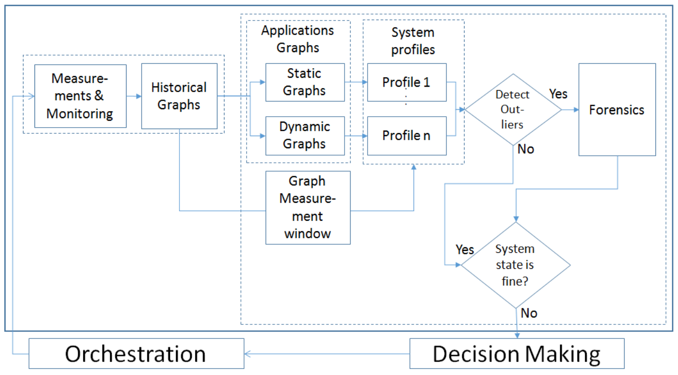

Fig. 1. Graph-Based Diagnosis In Software-Defined Infrastructure System Architecture

whether graph has become significantly different from its predecessors. Hence, it is necessary to define two things: first the features that represent a graph; second a distance measure between these features. Based on the distance, we can train the system to decide whether a specific graph is anomalous or not. Authors in [18] studied different graph similarity measures, anomaly detection techniques in large network based data and clustering similar graphs together. Different approaches have been used in detecting anomalies in dynamic graphs such as feature-based events [19] [20] [21], decomposition-based events [22] [23], clustering-based [24] [25] and window-based events [26] [27]. In [28], an eigen-space based approach has been proposed for modeling graphs and detecting anomalies.

In contrast to the current work, we focus on anomaly detection in the physical infrastructure itself, unlike [28], and detect a wider range of anomalies in several use cases. We have used a novel approach in detecting anomalies by leveraging both graph-mining metrics and machine learning techniques. Our work is the first to address graph-based anomaly detection in virtualized heterogeneous environments.

III. System Architecture

In this section of the paper we present our system architecture used for graph-based diagnosis in the SAVI testbed. Figure 1 depicts the architecture for our system. The system is composed of four main modules: The monitoring and measurements module, the diagnostics module, the decision making module and the orchestration module. The monitoring and measurements module is responsible for collecting different metrics from SAVI heterogeneous resources such as Network and Compute metrics and building graphs for different applications running in SAVI testbed. The diagnostics module is responsible for performing the Graph-based Anomaly Detection which present in this paper. The decision making module is responsible for performing suitable actions in order to heal the system from the effect of the anomalies. Finally, the orchestration module is responsible for executing the suitable decisions made in order to return the system to its steady state condition. The focus of this paper is on the diagnostics module as this where the anomaly detection is done.

IV. Graph Diagnosis Module Description

In this section of the paper we present our design for the graph-based diagnostics module of the system described in Section III.

A. Application Graphs

The Application Graphs module is responsible for identifying different applications graphs running in SAVI testbed. It is responsible for classifying application graphs into Static and Dynamic graphs. Since there are different anomaly detection techniques to be used, classifying application graphs into Static and Dynamic is important in identifying anomalies. The nature of distributed applications running in cloud platforms raise the importance of studying application graphs instead of individual resources behavior.

B. System Profiles

This module is responsible for saving different application profiles running in SAVI testbed. Those generated profiles represent the normal behavior state of the running applications. The profiles are made of different features and metrics calculated for different application graphs using NetworkX [29] software package. New incoming measurements are compared with those profiles in order to identify whether the monitored resources graphs are behaving in a normal manner or not.

C. Forensics

The Forensics module is responsible for investigating whether the detected graph anomalies resulted from an application misbehavior or not. Furthermore, the Forensics module is responsible for performing Root Cause Analysis for the detected graph anomalies. The Forensics module is responsible for investigating what are the sources of these graph anomalies as well as why these sources raise such anomalies to predict these anomalies in the future.

V. Proof of Concept

In this section of the paper, we demonstrate how our graph-based anomaly detection framework operates. The first 3 use cases illustrate static graphs scenarios while the last use case illustrates a dynamic graph one.

A. Webserver - Database workload pattern

In this scenario, we consider a workload running on a webserver that is serving requests by accessing a database as shown in Figure 2. When the workload increases on the webserver, the workload increases consequently on the database and vice-versa. In order to illustrate our graph-based anomaly detection approach, we have intentionally connected another webserver running a workload to the same database.

We train our system by monitoring the database behavior in two cases: First, running workload-1 as show in Fig. 2 while

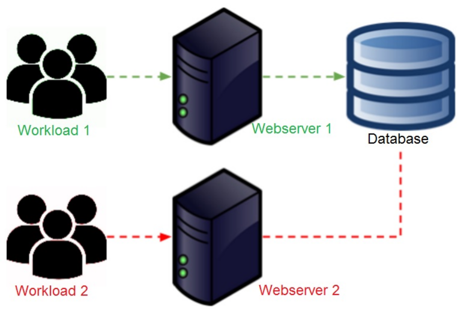

Fig. 2. Webserver - Database workload diagram

workload-2 is suspended. Afterwards, we suspend workload-1 and run workload-2.

The main idea in this scenario is to illustrate that monitoring the database application solely will not be able to detect anomalies as its pattern is periodic and normal looking. In order to detect anomalies in this scenario, we have used Linear Support Vector Classification (LinearSVC) [30] to perform the classification between normal behavior and anomalous behavior of the system. We have trained our system and we present our detection results in the evaluation section.

B. *Bandwidth throttling*

In this scenario, we consider two communicating virtual machines forming a graph. A virtual link between two Virtual Machines belonging to the same application is suffering from bandwidth throttling. This can be a useful scenario to detect the efficiency of isolation between different slices, as well as detecting misconfiguration of the network parameters.

In this scenario, we use time series adjacency matrices of the graph in order to detect anomalies. We calculate the distance between every two consecutive adjacency matrices. The distance $d_1(A,B)$ between adjacency matrices if the matrices are $A = (a_{ij})$ and $B = (b_{ij})$ could be expressed as : $d_1(A,B) = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} - b_{ij}$.

In order to train our system, first we initiate a file transfer between the two virtual machines and we calculate the different values of d1. Afterwards, we introduce a Bandwidth throttling over one of the virtual links between the two VMs and calculate the corresponding values of d1. Finally, we use LinearSVC in order to build a model. This model will be used in detecting Bandwidth throttling anomalies between the VMs in the evaluation section.

C. *DoS attack on a webserver*

In this scenario, we consider a graph composed of three nodes : Denial of Service attacking node, Webserver node and a back-end database node.

In this scenario, we use time series adjacency matrices of the graph in order to detect anomalies. We calculate the distance between every two consecutive adjacency matrices as previously discussed. In order to train our system, first we initiate a denial of service attack and we calculate the different values of d1. Afterwards, we use LinearSVC in order to build a model. This model will be used in detecting DoS anomalies in the evaluation section. The main difference between this case and the previous one is that the magnitude of d1 decreases in the Bandwidth throttling scenario whereas in the DoS scenario it increases.

D. *Spark Job failure*

In this scenario, we consider a graph of a Spark [31] Cluster composed of six nodes : A Spark Master node and five Spark worker nodes. The cluster that we are using is running a job of collecting monitoring data from SAVI testbed core node and saving them into Hadoop Distributed File System (HDFS) [32].

In this scenario, we use time series Assortativity coefficient [33] calculated for the graph in order to detect anomalies. In order to train our system, first we calculate the Assortativity coefficient for the Spark Cluster running the monitoring data collection job then we intentionally kill the job to generate the labeled training dataset. Afterwards, we use LinearSVC in order to build a model. This model will be used in detecting Spark Job failure anomalies in the evaluation section.

VI. Evaluation

In order to evaluate our system, we have conducted several experiments to verify our approach in detecting anomalies. We performed our experiments in the core node of the SAVI testbed, composed of over 20 physical servers hosting a few hundred VMs. We use the OpenStack and OpenFlow to collect data about the various elements in our network. Collected metrics include: CPU utilization, amount of disk read and write data, amount of memory read and write data, and network bandwidth between each pair of VMs. Our experiments are reproducible by requesting access to SAVI testbed from [34]. The details about the metrics available from Openstack can be found in [35]. We use Hadoop as our distributed file-system for data storage and Spark as our analytics framework. In the following subsections, we present the verification for each use case described in the Proof of Concept section.

1) Webserver - Database workload pattern: We have trained our system using 5 hours of data. Afterwards, we tested our system using 1.5 hour of data. We used the CPU Utilization metrics collected from the Webserver and Database. Figure 3 shows the testing phase of our system. The dash-doted curve represents the CPU Utilization of the Webserver, the dashed curve represents the CPU utilization of the Database. The solid curve represents the labels of the test data that we know beforehand ; high means anomaly, low means normal behavior. The dots represent the predicted labels of the data using LinearSVC. Our system was able to detect the 11 anomalies accurately as shown in Figure 3.

2) Bandwidth throttling: We have setup two Virtual Machines with xlarge flavor that has 160GB of disk. We have initiated a file transfer operation between the two VMs. The file size was 100GB with a transfer rate 10 Mbps between the two VMs.The throttling value in the training phase was fixed at 512 kbps. The training set for LinearSVC was 134 data points. Afterwards, we tested our system by repeating the

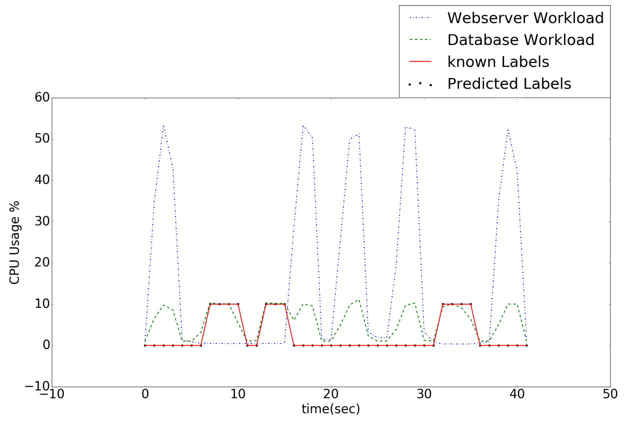

Fig. 3. Webserver Database testing phase

same experiment but while having random varying throttling value between 1 Mbps and 5 Mbps as shown in Figure 4. The dotted curve represents the time varying d1, the solid curve represents the labels of the test data that we know beforehand and the dots represent the predicted labels of the data using LinearSVC. Our system was able to detect the 35 anomalies accurately as shown in Figure 4.

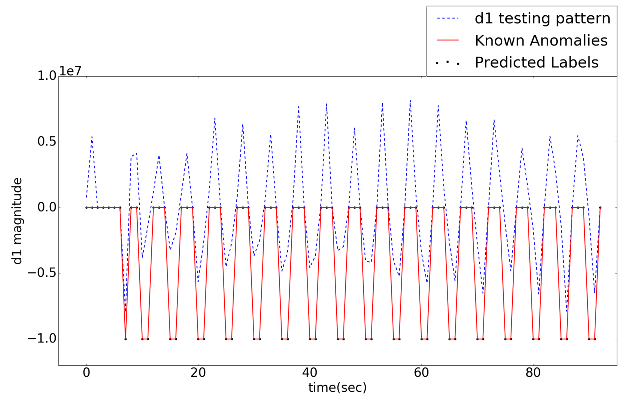

Fig. 4. Bandwidth throttling testing phase

3) Denial of Service Attack: We have trained our system using 118 data points by initiating a Denial of Service attack for 4.46 hours. Afterwards, we have tested our system by repeating the experiment for 2 hours as shown in Figure 5. The dashed curve represents the time varying d1, the solid curve represents the labels of the test data that we know beforehand and the dots represent the predicted labels of the data using LinearSVC. Our system was able to detect the 26 anomalies accurately as shown in Figure 5.

4) Spark Job Failure: We have trained our system by using the 327 data points collected from the Spark Cluster in 10.9 hours. Afterwards, we have tested our system by repeating the experiment for 5.5 hours as shown in Figure 6. The solid curve represents the time varying Assortativity Coefficient, the dashed curve represents the labels of the test data that we know beforehand. The dots represent the predicted labels of the data using LinearSVC; high means normal behavior, low means an anomaly. Our system was able to detect the 30 anomalies accurately as shown in Figure 6.

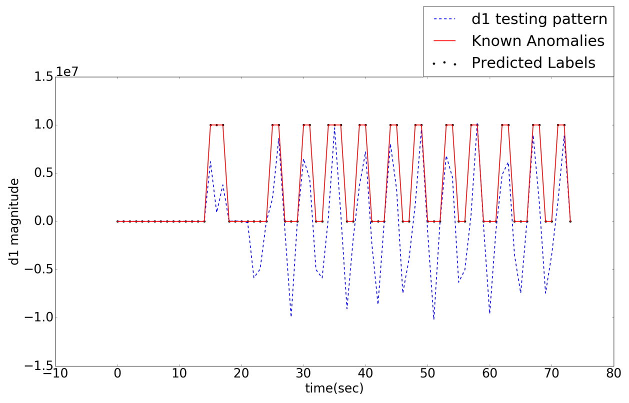

Fig. 5. DoS attack testing phase

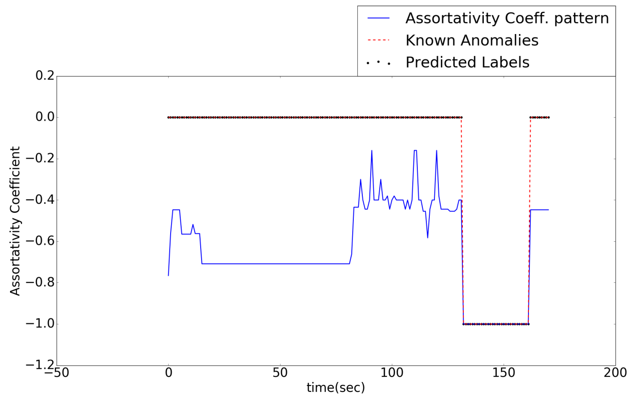

Fig. 6. Spark Job failure testing phase

We have repeated the previous experiments several times. Since the dimensionality of the data is relatively low as well as the linear separability nature of our problem the LinearSVC algorithm works accurately in all iterations. However, if the problem complexity increases the performance of the Support Vector Machines algorithm is expected to degrade and several simulations will be required to evaluate its performance. Dimensionality and non linear separable anomaly data can increase the problem complexity.

VII. Conclusion

In this paper we have designed and evaluated a graph-based diagnosis framework in Software-Defined Infrastructure running in SAVI testbed. Our framework is able to accurately detect system anomalies by leveraging different Graph-mining and Machine Learning techniques. We have tested our framework on several use cases covering different kinds of anomalies affecting various types of application graphs.

VIII. Future Work

Our future work includes extending our experiments by running our framework on many more test cases, many more times and provide accuracy measures such as false alarm probabilities.

References

[1] Joon-Myung Kang, Hadi Bannazadeh, and Alberto Leon-Garcia. Savi testbed: Control and management of converged virtual ict resources. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 664–667. IEEE, 2013.

[2] Available : https://www.openstack.org/.

[3] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[4] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data Mining and Knowledge Discovery*, 29(3):626–688, 2015.

[5] Keith Henderson, Tina Eliassi-Rad, Christos Faloutsos, Leman Akoglu, Lei Li, Koji Maruhashi, B Aditya Prakash, and Hanghang Tong. Metric forensics: a multi-level approach for mining volatile graphs. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 163–172. ACM, 2010.

[6] Phillip Bonacich and Paulette Lloyd. Eigenvector-like measures of centrality for asymmetric relations. *Social networks*, 23(3):191–201, 2001.

[7] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[8] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1231–1239. ACM, 2012.

[9] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[10] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *Knowledge Discovery in Databases: PKDD 2004*, pages 112–124. Springer, 2004.

[11] Hanghang Tong and Ching-Yung Lin. Non-negative residual matrix factorization with application to graph anomaly detection. In *SDM*, pages 143–153. SIAM, 2011.

[12] Xiaowei Xu, Nurcan Yuruk, Zhidan Feng, and Thomas AJ Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833. ACM, 2007.

[13] Caleb C Noble and Diane J Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2003.

[14] William Eberle and Lawrence Holder. Discovering structural anomalies in graph-based data. In *Data Mining Workshops, 2007. ICDM Workshops 2007. Seventh IEEE International Conference on*, pages 393–398. IEEE, 2007.

[15] Chao Liu, Xifeng Yan, Hwanjo Yu, Jiawei Han, and S Yu Philip. Mining behavior graphs for" backtrace" of noncrashing bugs. In *SDM*, pages 286–297. SIAM, 2005.

[16] Jing Gao, Feng Liang, Wei Fan, Chi Wang, Yizhou Sun, and Jiawei Han. On community outliers and their efficient detection in information networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 813–822. ACM, 2010.

[17] Emmanuel Müller, Patricia Iglesias Sánchez, Yvonne Mülle, and Klemens Böhm. Ranking outlier nodes in subspaces of attributed graphs. In *Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on*, pages 216–222. IEEE, 2013.

[18] Cemal Cagatay Bilgin and Bülent Yener. Dynamic network evolution: Models, clustering, anomaly detection. *IEEE Networks*, 2006.

[19] U Kang, Spiros Papadimitriou, Jimeng Sun, and Hanghang Tong. Centralities in large networks: Algorithms and observations. In *SDM*, volume 2011, pages 119–130. SIAM, 2011.

[20] Peter Shoubridge, Miro Kraetzl, WAL WALLIS, and Horst Bunke. Detection of abnormal change in a time series of graphs. *Journal of Interconnection Networks*, 3(01n02):85–101, 2002.

[21] Horst Bunke, Peter J Dickinson, Miro Kraetzl, and Walter D Wallis. *A graph-theoretic approach to enterprise network dynamics*, volume 24. Springer Science & Business Media, 2007.

[22] Leman Akoglu and Christos Faloutsos. Event detection in time series of mobile communication graphs. In *Army Science Conference*, pages 77–79, 2010.

[23] Ryan A Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 667–676. ACM, 2013.

[24] George Karypis and Vipin Kumar. Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[25] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[26] Carey E Priebe, John M Conroy, David J Marchette, and Youngser Park. Scan statistics on enron graphs. *Computational & Mathematical Organization Theory*, 11(3):229–247, 2005.

[27] Misael Mongiovi, Petko Bogdanov, Razvan Ranca, Ambuj K Singh, Evangelos E Papalexakis, and Christos Faloutsos. Netspot: Spotting significant anomalous regions on dynamic networks. In *Proceedings of the 13th SIAM international conference on data mining (SDM), Texas-Austin, TX*. SIAM, 2013.

[28] Tsuyoshi ID and Hisashi KASHIMA. Eigenspace-based anomaly detection in computer systems. In *tenth ACM SIGKDD international conference on Knowledge discovery and data mining KDD*, 2004.

[29] A Hagberg, D Schult, and P Swart. Networkx. *URL http://networkx. github. io/index. html*, 2013.

[30] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification. 2003.

[31] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[32] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[33] Rogier Noldus and Piet Van Mieghem. Assortativity in complex networks. *Journal of Complex Networks*, page cnv005, 2015.

[34] Request to access savi: http://www.savinetwork.ca/about-savi/request-access-to-savi-testbed/.

[35] OpenStack Meter description. http://docs.openstack.org/admin-guide/telemetry-measurements.html.