Executing Online Anomaly Detection in Complex Dynamic Systems

Tommaso Zoppi
Department of Mathematics and Informatics, University of Florence
Viale Morgagni 65, Florence, Italy
tommaso.zoppiunifi.it

***Abstract*—Revealing anomalies in data usually suggests significant - also critical - actionable information in a wide variety of application domains. Anomaly detection can support dependability monitoring when traditional detection mechanisms e.g., based on event logs, probes and heartbeats, are considered inadequate or not applicable. On the other hand, checking the behavior of complex and dynamic system it is not trivial, since the notion of "normal" – and, consequently, anomalous - behavior is changing frequently according to the characteristics of such system. In such a context, performing anomaly detection calls for dedicate strategies and techniques that are not consolidated in the state-of-the-art. The paper expands the context, the challenges and the work done so far in association with our current research direction. The aim is to highlight the challenges and the future works that the PhD student tackled and will tackle in the next years.**

Keywords—anomaly detection; monitoring; multi-layer; dynamicity; complex system; online

I. Introduction

Using anomaly detectors to assess the behavior of a target complex system at runtime is a promising approach that was explored in the last decade [1], [2]. Previous works showed that anomaly detection is a very flexible technique, analyzing different monitored behavioral data flows, finally allowing correlation between different events. This technique is commonly used to build error detectors [5], intrusion detectors [4] or failure predictors [3], assuming that a manifestation of an error or an adversarial attacker activity leads to an increasingly unstable performance-related behavior before escalating into a (catastrophic) failure. Anomaly detectors are in charge of i) detecting these fluctuations, and ii) alerting the administrator – who triggers proactive recovery or dumps critical data - with a sufficient look-ahead window. As stated in [1], anomaly detection strictly depends on the ability of distinguishing among normal and anomalous behavior. Unfortunately, complex and dynamic systems can often hide behavioral details (e.g., *Off-The-Shelf* components) or call for frequent reconfigurations, negatively affecting our ability in performing anomaly detection.

In particular, enterprise solutions such as *Nagios*, *Ganglia* or *Zenoss* allow the administrator to choose which indicators (e.g., CPU usage, accesses to hard disk) to observe, tracing their evolution through time. These enterprise tools also give the chance to setup static thresholds for each indicator, testing the availability of each single functionality or service exposed by the target system. Nevertheless, as expanded in Section III, they i) do not implement dynamic thresholds (e.g., statistical) for the monitored indicator, and ii) cannot easily adapt their behaviour when the configuration of the target system changes, calling for manual reconfigurations. This represents a strong limitation for the usage of such tools in dynamic systems.

The paper is structured as follows: Section II reports on anomaly detection, while Section III points out the motivations of our work and the related challenges. Section IV describes the framework for anomaly detection that is currently under investigation, while Section V and Section VI conclude the paper focusing on the ongoing and planned future works.

II. Anomaly Detection in Complex Dynamic Systems

Dynamicity is the property of an entity that constantly changes in term of offered services, built-in structure and interactions with other entities. From one side, this defines systems that can adapt their behavior according to the actual needs, but on the other side it makes all the behavioral checks harder to execute since the normal behavior is changing very often. Regarding dependability assessment, this means that failure predictors must be kept up-to-date as the system is running, calling for a new training phase aimed at reconfiguring all the involved parameters. This calls for a monitoring solution that i) continuously observes the system to avoid or mitigate failures of attack, ii) gathers data from modules or layers where possible, and iii) is able to infer the status of the whole system looking *only* at data collected at its constituent modules. It follows that detection algorithms based on fingerprints e.g., antiviruses [14], intrusion detectors [13] or failure predictors [9], may result not adequate for complex systems due to their intrinsic dynamicity.

In such a context, anomaly detection seems one of the most suitable approaches in detecting unexpected behaviors in dynamic and complex systems. In the security domain, this technique was proven effective [14] in detecting zero-day attacks, which exploit unknown vulnerabilities to get into the targeted system. Antiviruses and intrusion detectors can detect hazards when they identify a behavior that is compliant with a known fingerprint of an attacker or a malware, but they need also rules to detect zero-day attacks or attacks from unknown adversaries [12]. The same approach is commonly used to detect threats to dependability in complex systems, also when the system is composed by OTS components [10], [7]. Moreover, unexpected or previously unknown system failures can be predicted observing specific indicators to characterize if the runtime system behavior is compliant with given performance expectations [9], [10]. Several frameworks targeting anomaly

detection in a specific subset of complex systems, namely Systems-of-Systems (SoSs) are summarized in [20]. More in detail, some of them deal with dynamicity [11], while others tackle systems composed of OTS components [9], [10]. All the considered frameworks are realized either for dependability [9], [10], [11] or security [4], [12] purposes.

III. Main Challenges

To the authors' knowledge, the topic of bringing anomaly detection into the design of complex dynamic systems e.g., Service Oriented Architectures (SOAs) [8] or SoSs [20], was not explored in the recent years. Consequently, after expanding the topic of *anomaly detection*, in the rest of the paper we will report on both the motivations of the research and the challenges that the 3rd-year PhD student tackled in the first two years, with a closer look to the next planned research steps.

Summarizing, the tackled research challenges are:

CH1. *Design a monitoring and anomaly detection framework that is suitable for dynamic and/or distributed systems* and therefore is tailored to automatically adapt its parameters depending on the current configuration of the target system;

CH2. Provide a *flexible monitoring* strategy that copes with dynamicity of complex systems. The strategy must allow the collection of data coming from different system layers and constituent machines that can be updated frequently;

CH3. Understand the expected behavior of the system according to its current context. The *context must be detected at runtime*, calling for specific training phases that should not interfere with the normal usage of the platform (e.g., availability to the users)

CH4. Analyze monitored data to *extract the minimum set of features* (i.e., anomaly checkers and, consequently, monitored indicators) which guarantees the best tradeoff between monitoring effort and efficiency of the anomaly detection process at runtime;

IV. Building a Framework for Multi-Layer Anomaly Detection in Complex Dynamic Systems

A. Designing the framework

In [7] the authors applied the *Statistical Predictor and Safety Margin* (SPS) algorithm to detect the activation of software faults in an *Air Traffic Management* (ATM) system, that has few defined functionalities with respect to a SOA. Observing only *Operating System* (OS) indicators, SPS allowed performing error detection with high precision. This is a promising approach since it i) relies on a multi-layer monitoring strategy that allows to infer the state of the applications looking only at the underlying system layers (see CH2), and ii) uses a sliding-window-based machine learning algorithm that automatically tunes its parameters as the data flows (see CH1). Therefore we adapted this approach to work in a more dynamic context [5] where we instantiated the framework on one of the 4 virtual machines running the prototype of the Secure! [6] SOA.

The results achieved showed that analysing such a dynamic system without adequate knowledge on its behavior reduces the efficiency of the whole solution. We explain these outcomes as follows. SPS detects changes in a stream of observations identifying variations with respect to a predicted trend: when an observation does not comply with the predicted trend, an alert is raised. If the system has high dynamicity due to frequent changes or updates of the system components, or due to variations of user behaviour or workload, such trend may be difficult to identify and thus predict. Consequently, our ability in identifying anomalies is affected because boundaries between normal and anomalous behaviour cannot be defined properly.

Consequently, we investigated which information on SOA services we can obtain in absence of details on the services internals and without requiring user context (i.e., user profile, user location). In SOAs, the different services share common information through an *Enterprise Service Bus* (*ESB*, [8]) that is in charge of i) integrating and standardizing common functionalities, and ii) collecting data about the services. This means that static (e.g., services description available in *Service Level Agreements* - SLAs) or runtime (e.g., the time instant a service is requested or replies, or the expected resources usage) information about the context can be retrieved using knowledge given by ESB. In particular, having access to the ESB provides knowledge on the set of generic services running at any time *t*. We refer to this information as *context-awareness* of the considered SOA.

We can exploit this information to define more precisely the boundaries between normal and anomalous behaviour of the system under observation. For example, consider a user that invokes a *store file* service at time *t*. We can combine context-awareness with information on the usual behaviour of the service, which here regards data transfer. Therefore, if the *store file* service is invoked at time *t*, we expect the exchange of data during almost the entire execution of the service. Contrary, we can reveal that something anomalous is happening. This also highlights that the observation of lower levels make us able to identify the manifestation of errors at service level, ultimately providing both i) monitoring flexibility, and ii) maximum detection capability.

B. High-Level Architecture

In Figure 1 we depicted a high-level view of the framework. Starting from the upper left part of the figure, the framework can be described as follows. The user executes a *workload*, which is a sequence of invocations of SOA services hosted on the *Target Machine*. In this machine, *probes* are running, observing the indicators coming from 3 different system layers: i) *OS*, ii) *middleware* and iii) *network*. These probes collect data, providing a *snapshot* of the target system composed by the observation of indicators retrieved at a defined time instant. The probes forward the snapshot to the *communication handler*, which encapsulates and sends the snapshot to the other *communication handler*. Data is analyzed on a separate machine, the *Detector Machine*. This allows i) not being intrusive on the Target Machine, and ii) connecting more Target Machines to the same Detector Machine (note that the number of Target Machines is limited by the computational resources of the Detector Machine). The communication handler of the Detector Machine collects and sends these data to the *monitor aggregator*, which merges them with *runtime information* on the

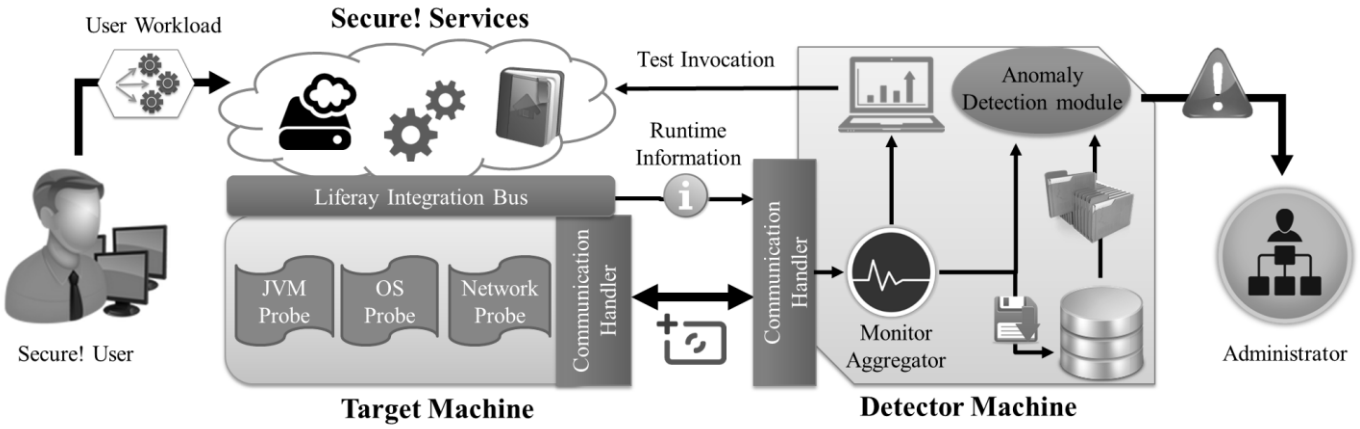

Fig 1. High-level view of the multi-layer anomaly detection framework

context (e.g., the sequence of service calls) obtained from the ESB. Looking at *runtime information*, the monitor aggregator can detect changes in the SOA and notify the administrator that up-to-date services information is needed to appropriately tune the anomaly detector. The administrator is in charge of running tests (*test invocation*) to gather novel information on such services.

The snapshots collected when the SOA is opened to users are sent to the *anomaly detection module*, which can query the database for contextual services information. The *anomaly detection module* analyzes each observed snapshot to detect anomalies. If an anomaly is detected, an alert is raised to the *system administrator* that takes countermeasures and applies reaction strategies. These are outside from the scope of this work and will not be elaborated further.

C. Exercising the framework

The framework is instantiated specifying: i) a *workload*, ii) the approach to obtain contextual data (as mentioned in Section 4.A), iii) the set of monitored indicators, and iv) the amount of data needed for training. The methodology is composed of two phases: *Training Phase* and *Runtime Execution*.

Training Phase. First, the approach (static or runtime) to obtain *contextual data* characterizing the fingerprint of the investigated services is instantiated. Then, training data is collected during the execution of the chosen workload, storing in a database a time series for each monitored indicator representing the evolution of its value during the train experiment. These data are complemented with data collected conducting anomaly injection campaigns, where anomalies are injected in one of the SOA services, to witness the behavior of the target system in such situations. These data are lastly used by the *anomaly detection module* to tune its parameters depending on the current context. Injected anomalies simulate the effect of the manifestation of an error or of an upcoming failure, e.g., anomalous resource usage.

Runtime Execution. Once the training phase is ended and the parameters of the anomaly detector are defined, the system is opened to users. *Monitor aggregator* merges each snapshot observed by the *probing system* with *runtime information*, and it sends them to the *anomaly detection module*. This module provides a numeric anomaly score: if the score reaches a specified threshold, an anomaly alert is risen and the administrator is notified. If during this phase a change in the system is detected, a new *training phase* is scheduled and it will be executed depending on the policies defined by the administrator (e.g., during lunchtime, instantly, at night).

V. Dealing with Online Training

According to the description in Section 4, we can observe how the availability of the anomaly detector is affected from the time it needs to train its algorithms and to detect the contextual data (see CH3). All the detection systems that depend on training data have to deal with this turnover between training phase – in which the system is tuning the detector – and runtime execution – where the system is opened to users and executing its tasks with anomaly detection support.

A. Limitations of Training Phases

The time requested by the training phase is considered not influent in systems that i) can be put offline in defined time periods (e.g., servers that are unused at night), or ii) have static behaviors (e.g., air traffic management systems), meaning that the training phase can be executed once keeping their results valid through time. Nevertheless, a big subset of systems do not adhere with these specifications since they have a dynamic behavior that calls for frequent training phases needed to adapt the parameters of the anomaly detector to the current context. In such a context, frequent training phases are needed to keep the anomaly detection logic compliant with the current notion of "normal" and "anomalous" behavior. Unfortunately, during these training phases the anomaly detector is working with outdated parameters negatively affecting the correct detection of anomalies. To limit these adverse effects, several authors [16], [17], [18], [19] dealing with detector or predictors in the context of complex systems proposed an "online training" approach.

B. Online Training

A strong support for the design of online training techniques comes from systems that study trajectories [15], [16]. In this field, abnormal trajectories tend to carry critical information of potential problems that require immediate attention and need to be resolved at an early stage [15]. The trajectories are continuously monitored as they evolve to understand if they are following normal or anomalous paths, ultimately breaking the classic training-validation turnover (see *conformal anomaly*

detection [16]). In [17], authors tackle online training for failure prediction purposes i) continuously increasing the training set during the system operation, and ii) dynamically modifying the rules of failure patterns by tracing prediction accuracy at runtime. A similar approach is adopted also to model up-to-date *Finite State Automata* tailored on sequences of system calls for anomaly-based intrusion detection purposes [18] or *Hidden Semi Markov Models* targeting online failure prediction [19].

C. Shaping Online Training for Dynamic Systems

When the target system is dynamic, it can change its behavior in different ways, consequently affecting the notion of normal or expected behavior. These changes must trigger new training phases aimed at defining the "new" normal behavior, allowing the parameters of the anomaly detector to be tailored on the current version of the system. Moreover, according to [17], the training set is continuously enriched by the data collected during the executions of services, providing wide and updated datasets that can be used for training purposes. This training phase starts once one of the triggers will activate, calling for complex data analytics that are executed on a dedicated machine, to do not bother the target system with these heavy computations.

Looking at the possible ways systems have to dynamically change their behavior, we are currently considering as triggers: i) *update of the workload,* ii) *addition or update of a service* in the platform, iii) *hardware update*, and iv) *degradation of the detection scores.* While the first three triggers can be detected easily either looking at the basic setup of the SOA or after a notification of the administrator, the degradation of detection scores needs more attention. Concisely, the dynamicity of the system is not only due to events that can alter its behavior (i.e., the first three triggers). The notion of normal behavior may be affected also by changes of the environment or of some internals of the systems than cannot be easily identified. Unfortunately, they might lead to a performance degradation of the anomaly detector (e.g., higher number of false positives) ultimately calling for an additional training phase.

VI. Conclusions and Future Works

This paper presents the topic the student is tackling during his PhD. More in detail, after describing the research area and the state of the art that was consolidated in the recent years, the paper addressed the motivations and the related challenges. Then, we described a framework for multi-layer anomaly detection in complex and dynamic systems that we developed during the PhD research period.

Future works will be oriented to deal with the dynamicity of complex systems. While a strategy for the collection and the analysis of data is already implemented in the framework mentioned before, some improvements need to be considered in order to make this framework suitable for dynamic systems. In particular, we will go through the online training approach, looking for strategies that will allow having always a clear definition of normal behavior. As discussed in Section 5.C, this will require deeper investigations on all the possible ways systems have to dynamically change their behavior. Moreover, we are planning to tackle the feature selection problem (see CH4) after the dynamic characteristics will be clearly defined.

Acknowledgment

This work has been partially supported by the IRENE JPI Urban Europe, the EU-FP7-ICT-2013-10-610535 AMADEOS and the EU-FP7-IRSES DEVASSES projects.

References

[1] Chandola, Varun, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." ACM computing surveys (CSUR) 41.3 (2009): 15.

[2] Rajasegarar, Sutharshan, Christopher Leckie, and Marimuthu Palaniswami. "Anomaly detection in wireless sensor networks." IEEE Wireless Communications 15.4 (2008): 34-40.

[3] Özçelik, Burcu, and Cemal Yılmaz. "Seer: a lightweight online failure prediction approach." IEEE Transactions on Software Engineering (2013).

[4] Salama, Shaimaa Ezzat, et al. "Web anomaly misuse intrusion detection framework for SQL injection detection." Editorial Preface 3.3 (2012).

[5] Ceccarelli, Andrea, et al. "A multi-layer anomaly detector for dynamic service-based systems." Int. Conference on Computer Safety, Reliability and Security (pp. 166-180). Springer Int. Publishing, SAFECOMP 2015.

[6] Secure! project, http://secure.eng.it/ (last accessed 1st December 2016)

[7] Bovenzi, Antonio, et al. "An OS-level Framework for Anomaly Detection in Complex Software Systems." Dependable and Secure Computing, IEEE Transactions on 12.3 (2015): 366-372.

[8] Erl, Thomas. Soa: principles of service design. Vol. 1. Upper Saddle River: Prentice Hall, 2008.

[9] Baldoni, Roberto, Luca Montanari, and Marco Rizzuto. "On-line failure prediction in safety-critical systems." Future Generation Computer Systems 45 (2015): 123-132.

[10] Williams, Andrew W., Soila M. Pertet, and Priya Narasimhan. "Tiresias: Black-box failure prediction in distributed systems." Parallel and Distributed Processing Symposium, IEEE 2007 (pp. 1-8). IPDPS 2007.

[11] Zoppi, Tommaso, Andrea Ceccarelli, and Andrea Bondavalli. "Context-Awareness to Improve Anomaly Detection in Dynamic Service Oriented Architectures." International Conference on Computer Safety, Reliability, and Security (pp 145-158). Springer International Publishing, 2016.

[12] Perdisci, Roberto, et al. "McPAD: A multiple classifier system for accurate payload-based anomaly detection." Computer Networks 53.6 (2009): 864-881.

[13] Mukkamala, Srinivas, Guadalupe Janoski, and Andrew Sung. "Intrusion detection using neural networks and support vector machines." Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on. Vol. 2. IEEE, 2002.

[14] Comar, Prakash Mandayam, et al. "Combining supervised and unsupervised learning for zero-day malware detection." INFOCOM, 2013 Proceedings IEEE (pp. 2022-2030). IEEE, 2013.

[15] Bu, Y., Chen, L., Fu, A. W. C., & Liu, D. (2009, June). Efficient anomaly monitoring over moving object trajectory streams. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 159-168). ACM.

[16] Laxhammar, Rikard, and Göran Falkman. "Online learning and sequential anomaly detection in trajectories." IEEE transactions on pattern analysis and machine intelligence 36.6 (2014): 1158-1173.

[17] Gu, J., Zheng, Z., Lan, Z., White, J., Hocks, E., & Park, B. H. (2008, September). Dynamic meta-learning for failure prediction in large-scale systems: A case study. In 2008 37th International Conference on Parallel Processing (pp. 157-164). IEEE.

[18] Sekar, R., Bendre, M., Dhurjati, D., & Bollineni, P. (2001). A fast automaton-based method for detecting anomalous program behaviors. In Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on (pp. 144-155). IEEE.

[19] Salfner, F., & Malek, M. (2007, October). Using hidden semi-Markov models for effective online failure prediction. In Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE Int. Symp. on (pp. 161-174). IEEE.

[20] Zoppi, Tommaso, Andrea Ceccarelli, and Andrea Bondavalli. " Exploring Anomaly Detection in Systems of Systems." To Appear at Symposium on Applied Computing (SAC) 2017, Marrakesh, Morocco.