Challenging Anomaly Detection in Complex Dynamic Systems

Tommaso Zoppi, Andrea Ceccarelli, Andrea Bondavalli
Department of Mathematics and Informatics, University of Florence
Viale Morgagni 65, Florence, Italy
{tommaso.zoppi, andrea.ceccarelli, andrea.bondavalli}@unifi.it

Abstract—**Software infrastructures are becoming more and more complex, making performance and dependability monitoring in wide and dynamic contexts such as Distributed Systems, Systems of Systems (SoS) and Cloud environments an unachievable goal. Consequently, it is very difficult to know how all the specific parts, services and modules of these systems behave. This negatively impacts our ability in detecting anomalies, because the boundaries between normal and anomalous behaviors are not always known. The paper describes the context and the targeted problem highlighting the research directions that the student will follow in the next years. In particular, after introducing the relevance of this work with respect to the academic and the industrial state of the art, we carefully define the problem and summarize the main challenges that arise according to such problem definition.**

Keywords—anomaly detection; monitoring; multi-layer; distributed system; complex system;

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 years [1], [3]. 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 [8], intrusion detectors [5] or failure predictors [4], 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 dump 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, to the best of authors' knowledge, complex and dynamic systems can often hide behavioral details (e.g., *Off-The-Shelf* components) or call for frequent reconfigurations, negatively affecting the detection.

However, it is possible to find well-known monitoring tools that can also check if the observed behaviour is compliant with the expectations [6]. In particular, enterprise solutions such as *Nagios*, *Ganglia* or *Zenoss* allow the administrator to choose which system indicators (e.g., CPU usage, HD accesses) to observe, tracing their evolution through time. These industrial 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.

II. Problem Definition

Let us consider a generic complex system composed of a set of machines M, where each machine has its computational and storage power, eventually unified through *Distributed File System* (*Distributed Systems*) or *Enterprise Service Bus* (e.g., *Service Oriented Architectures*). To enhance and support the runtime verification capabilities of such system we can include an anomaly detection framework that is composed by i) the *monitoring module*, and ii) the *anomaly detector*.

The *monitoring module* is a triple <*Mm*, *ind*, *rate*> where *Mm* specifies the monitored machines (Mm $\subseteq$ M) and the function *ind*: M $\rightarrow 2^I$ defines the subset of system indicators I (e.g., {*RAM usage*, *CPU usage*, *TCP sockets*} $\subseteq$ I) that will be monitored for a specific machine *m* ϵ M. *Rate* specifies the interval between two subsequent observation of system indicators by the probes. Consequent observation of the values of the same indicator constitutes an *indicator data series*: several indicator data series can be combined using arithmetic operators obtaining *complex data series* (e.g., the complex data series *cache_hit_rate* can be obtained as *cache_hits* / *cache_misses*). The monitoring activity provides a set $S = \{s_k = \{<i, m, ov> \mid i \in I, m \in Mm, ov \in \mathbb{R}\} \mid k \text{ in } K\}$ of *snapshots* that will be analyzed by the anomaly detector. Each snapshot contains a list of values *ov* of an indicator *i* observed in one of the monitored machines *m* at a defined time instant. Elements of complex data series can be obtained for each snapshot taking the respective indicator values and combining them following specific arithmetic rules.

The *anomaly detector* <SC, s, vs> evaluates a snapshot s ϵ S through a set of selected anomaly checkers SC ($\subseteq$AC) considering vs ϵ VS as voting strategy. An *anomaly checker* c ϵ AC represents the instantiation of an anomaly detection algorithm on a specific data series ds ϵ DS. For example, it can represent the application of a machine-learning algorithm on the *cache_hit_rate* complex data series, or on the *RAM_usage* indicator data series. Note that only past snapshots are used to tune the algorithms, which therefore evaluate the each *ds* value depending on the values obtained in previous snapshots. Each anomaly checker runs its algorithm and replies with an *anomaly* *score* indicating the result of the execution of its anomaly detection algorithm. Once all the selected checkers c ϵ SC have calculated their anomaly scores, the *vs* strategy aggregates these results and provides the final anomaly score for the whole snapshot. For example, since the voting strategy aggregates results coming from different anomaly checkers, it can perform a (weighted) sum of the single anomaly scores (e.g., the snapshot is anomalous if the sum is over a defined threshold).

In Figure 1 we can observe an instantiation of the problem: the target system is composed by three machines: M_1 and M_2 are connected through the same LAN, while M_3 is a cloud server reachable through the Internet. The monitor module observes two machines Mm = {M_1, M_3} and, consequently, two sets of indicators *ind*(M_1)={*RAM_usage*} and *ind*(M_3)={*cache_hits*, *cache_misses, CPU_Usage*}. Once per second, the monitor module builds a snapshot of the target system observing the indicators in *ind*(M_1) ∪ *ind*(M_3). The anomaly detector runs three checkers (|SC| = 3): a statistical check (i.e., regression) on *RAM_Usage*@M_1 and *CPU_Usage*@M_3 and a machine-learning algorithm (i.e., clustering) using the complex series *cache_hit_rate*@M_3. The snapshot is evaluated as anomalous if at least one of the three anomaly checkers raises an anomaly.

III. Main Challenges

Summarizing, the tackled research challenges are:

CH1. *infer the behavior of applications and services looking only at the underlying layers*, with a flexible monitoring infrastructure that allows the collection of data coming from different system layers and constituent machines;

CH2. *design an 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;

CH3. analyze monitored data to *extract the set of features* (i.e., anomaly checkers and, consequently, monitored indicators) which guarantees the best tradeoff between amount of monitored data and efficiency of the anomaly detection process at runtime;

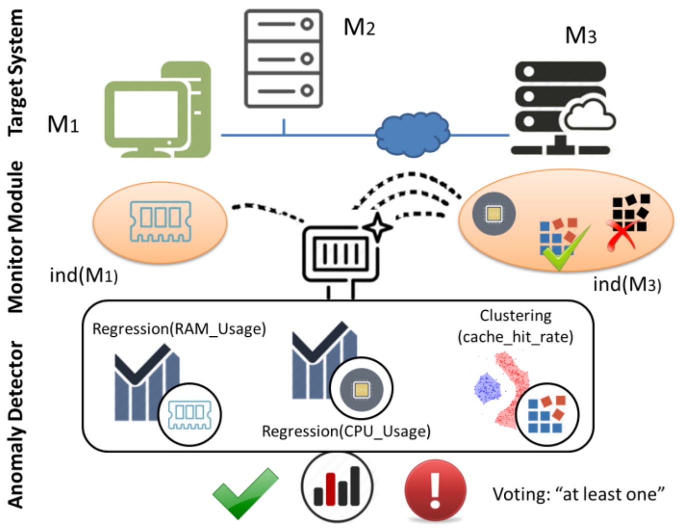

Fig. 1. Possible instantiation of the problem.

IV. Research Directions

During the first year of his PhD, the student developed a multi-layer monitoring framework for anomaly detection (CH1 and CH2) that can be installed on the targeted system performing injections of anomalies to verify its effectiveness and reactivity in detecting the anomalies mentioned above. The framework [8], [9] aims at detecting anomalies due to the manifestation of i) software bugs, ii) resource exhaustion, and iii) configuration errors that damage synchronization or communication (e.g., deadlock). Experiments are conducted on a testbed composed by two machines: one hosts the target system and the other hosts both the monitor and the anomaly detector.

Future researches will be oriented to tailor the functionalities of the framework on a distributed system, considering the need [10] of a global synchronization to order the (anomalous) events monitored in such a distributed scenario. Starting from our previous work, we aim at i) enhance the monitoring capabilities, becoming able to monitor several machines constituting the distributed system simultaneously, and ii) expand the considered library of anomalies, including the ones that may emerge in a distributed system [7] (e.g., unwanted synchronization, trashing). This will allow to tackle CH3, extracting the features (i.e., anomaly checkers) that performs better in detecting anomalies, classifying which indicators and system layers are more relevant for anomaly detection in a distributed system.

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] Fu, Qiang, et al. "Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis." ICDM. Vol. 9. 2009.

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

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

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

[6] Zanikolas, Serafeim, Rizos Sakellariou. "A taxonomy of grid monitoring systems." Future Generation Computer Systems 21.1 (2005): 163-188.

[7] Mogul, Jeffrey. "Emergent (mis) behavior vs. complex software systems." ACM SIGOPS Operating Systems Review. Vol. 40. No. 4. ACM, 2006.

[8] 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.

[9] Zoppi, Tommaso, et al. "Context-Awareness to improve Anomaly Detection in Dynamic Service Oriented Architectures", to appear at Int. Conference on Computer Safety, Reliability, and Security. Springer Int. Publishing, SAFECOMP 2016.

[10] Kopetz, Hermann. Real-time systems: design principles for distributed embedded applications. Springer Science & Business Media, 2011.