Contextual Anomaly Detection for a Critical Industrial System based on Logs and Metrics

Mostafa Farshchi*†, Ingo Weber†*, Raffaele Della Corte‡, Antonio Pecchia‡, Marcello Cinque‡,
Jean-Guy Schneider*†, John Grundy§,
*Swinburne University of Technology, Melbourne, Australia. {*mfarshchi, jschneider*}*@swin.edu.au*
†Data61, CSIRO, Sydney, Australia. {*firstname.lastname*}*@data61.csiro.au*
‡Federico II University of Naples, Italy. {*raffaele.dellacorte2, antonio.pecchia, macinque*}*@unina.it*
§Monash University, Melbourne, Australia. *john.grundy@monash.edu.au*

***Abstract*—Recent advances in contextual anomaly detection attempt to combine resource metrics and event logs to uncover unexpected system behaviors at run-time. This is highly relevant for critical software systems, where monitoring is often mandated by international standards and guidelines. In this paper, we analyze the effectiveness of a metrics-logs contextual anomaly detection technique in a middleware for Air Traffic Control systems. Our study addresses the challenges of applying such techniques to a new case study with a dense volume of logs, and finer monitoring sampling rate. Guided by our experimental results, we propose and evaluate several actionable improvements, which include a change detection algorithm and the use of time windows on contextual anomaly detection.**

***Index Terms*—Anomaly detection, contextual anomaly, system monitoring, log analysis, change detection.**

I. Introduction

Anomaly detection is a core concern for dependability practitioners. Research trends in this area propose the detection of **contextual anomalies** (as opposed to *point* anomalies) [1], [2], [3], [4]. These combine resource utilization metrics (*e.g.*, CPU and memory utilization, network traffic) and additional contextual data with the aim of pinpointing unexpected system behaviors and malfunctions. Recent work attempts to combine resource metrics and *event logs* [5], [6] for contextual anomaly detection. This paper explores the use of contextual anomaly detection in a critical industrial system; we base the analysis on our previously proposed approach [6], [7], which is **non-intrusive** and **unsupervised**. The aim of this paper is twofold: (i) experimenting with a metrics-logs contextual anomaly detection method in a new compelling domain, and (ii) contributing real-world case study experience for this technique. The system is a middleware for the integration of Air Traffic Control (ATC) applications by a world-leading company in electronic and information technologies.

The key contributions of this work are: *1. An automated abstraction method to infer system activities from logs.* We adopt an automatic method to extract regular expressions and find unique event types across the logs of the reference system. Particularly, this addresses the limitation of applying the previous method [6] on dense logs of common application operation environments. *2. An assessment of the limitations of metrics-logs contextual anomaly detection*. We use a fault-injection approach to collect data records during system failures and to elicit stressful operations. Experiments reveal that a large difference between the predicted and actual resource utilization causes sporadic false positives. *3. Means for improving contextual anomaly detection*. Based on the results of our experimental analysis, we proposed, implemented, and evaluated means for improving anomaly detection, which include a (i) **change detection algorithm** for reducing false positives, and (ii) **time-window-based approach** to enlarge the observation period and increase detection accuracy.

A long version of this paper is available as a technical report [8], which provides additional details and discussions. The rest of the paper is organized as follows: we introduce our case study in Section II, followed by Section III that provides an overview of the approach. Assessments methods and improvement results are presented in Sections IV and V.

II. Case Study System

We use a **middleware platform** for the integration of mission-critical applications in the Air Traffic Control (ATC) domain [9]. The high-level architecture is shown in Fig. 1. It consists of the middleware and two ATC applications: (i) a Flight Data Processor (FDP), which generates/updates flight data and (ii) a web-based Controller Working Position (CWP), which receives the data from the middleware and presents it on a web console. The middleware consists of *transport* and *adaptation* layers. The transport layer ensures the communication between the FDP and CWP, according to the publish-subscribe paradigm. The adaptation layers allow applications to use the middleware and its services. The **monitoring layer** uses a Linux loadable kernel module (LKM) to collect resources usage data (e.g., CPU and RAM).

OS processes running the FDP and the middleware are monitored with a 100-millisecond sampling rate. The middleware generates dense and large volume of logs: for example, it can generate around 2700 lines during five minutes of operations and up to 240 lines per second. It should be noted that – although we deployed a controlled testbed for experimental purposes – the ATC *middleware*, *applications* and *test suite* consist of the real-world software made available by the industry provider. Log-events from the middleware and usage metrics from monitoring layer are used for our anomaly detection experiments.

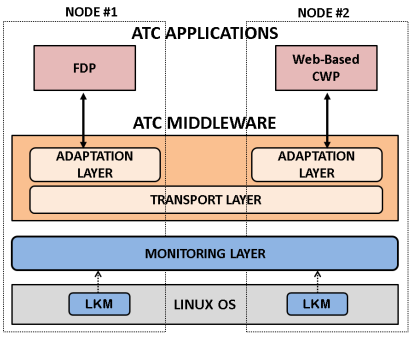


Fig. 1: High-level architecture of the reference system.

III. Overview of Analysis Method

The contextual anomaly detection technique proposed in [6] that we are building upon extracts a regression-based model that exploits correlation between events (as captured in system logs – *e.g.*, logs) and resource metrics (collected by monitoring services – *e.g.*, CPU usage).

The technique requires (i) a stream of time-stamped events, such as events recorded by log lines, representing the *behavioral context* of a system's operation, and (ii) at least one (preferably more) resource metric, representing the run-time "state" of a system. At a high-level view the approach requires to extract event types from log events and represent clusters of log event types as a set of quantitative metric. Once a quantitative metric of log events is available, an events-metric correlation model is derived using a suitable regression model (with event type occurrences as independent variables and a monitoring metric as dependent variable) for each of the monitoring metrics. This lead to identifying the predictability power for each of the generated events-metric correlation models and select the one(s) with the best "sensitivity" with regards to detecting changes in system metrics. Having the regression model, we derive the assertion specification from regression analysis, which we use for runtime monitoring.

Above steps are explained in details in our technical report [8] and further informtion can be found in [6].

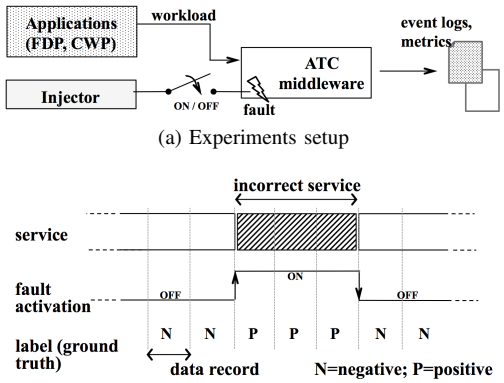
IV. Anomaly Detection with Initial Approach

We identify the most sensitive metric, i.e., *CPU usage* in this study, by means of the regression model and derive the following **assertion equation** to predict CPU usage from logs:

$$y_i = 1.462 + 4.606 * A06_i + ... + 3.557 * A17_i \quad (1)$$

We leverage this obtained model for our contextual anomaly detection approach: at each time t_i, the actual CPU usage value is expected to be predicted by the equation with an error of estimate of ± 6.059; otherwise, it is raised an anomaly.

It is important to note that we performed our analysis based on four separate runs of the target system. In the *normal* run (without any fault injection), we learned the model of the normative system behavior; we then validate the accuracy of the learned model with three further runs.



(a) Experiments setup

(b) Data record collection

Fig. 2: Fault injection setup and records collection

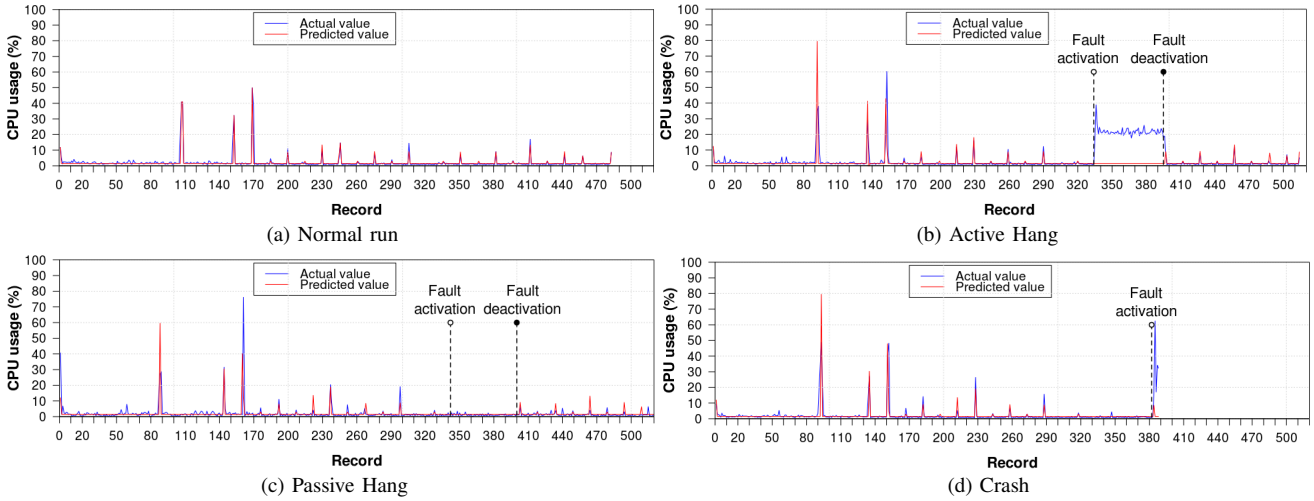
A. Data collection and evaluation metrics

Data are collected by running the reference system with three failure settings independently: **Active hang**: the system appears to be running, but its services are perceived as unresponsive. **Passive hang**: the system appears to be running; its services are perceived as unresponsive because the system is indefinitely waiting for an event to occur. **Crash**: the system terminates unexpectedly and no service is provided at all. The above-mentioned failures are based on a widely-accepted taxonomy in dependability research [10]. It should be noted that these failure settings are not meant to be exhaustive, but aim to elicit a number of stressful conditions requiring improvements for contextual anomaly detection.

Failures are induced by means of fault injection. To achieve this, the source code of the ATC middleware is arranged in order to allow an external component, called *Injector* in the following, to activate and deactivate software faults on demand, during the progression of the system execution.

Fig. 2a shows the experimental setup. Fig. 2b depicts the timing of the experiments. For each run, the ATC middleware and applications (FDP, CWP) are initialized and then the applications run the workload by the industry provider. The system is run regularly for several workload cycles until the Injector activates a software fault (*i.e. fault activation* switches from OFF to ON in Fig. 2b). Injection stays ON for one minute, which causes the system to deliver an incorrect service. In case of active/passive hang, the regular system function resumes after the injection (*i.e. fault activation* switches from ON to OFF); in case of crash, the system goes out of service a few seconds after the injection.

We collected data records throughout the system execution. Each data record contains the information of 1-second time windows: (i) the *actual* value of CPU usage, (ii) the *predicted* value of CPU usage based on the assertion Equation (1), (iii) the outcome of the detector based on the comparison of actual-predicted metric, and (iv) the *label* of the record. The **label** is *positive* (P) if the record is collected under incorrect service; *negative* (N) otherwise (Fig. 2b). We use the label as *ground truth*, or *oracle*, to assess the outcome of the anomaly detector based on the metrics of *precision*, *recall* and *accuracy*.



(a) Normal run (b) Active Hang

(c) Passive Hang (d) Crash

Fig. 3: *Actual* versus *predicted* CPU usage with highlighted fault activation periods.

TABLE I: Initial anomaly detection results

	Active Hang	Passive Hang	Crash
Total Records	513	519	387
True Negatives	443	448	372
False Negatives	0	60	0
True Positives	62	0	4
False Positives	8	11	11
Precision	0.886	0.000	0.267
Recall	1.000	0.000	1.000
Accuracy	0.984	0.863	0.972

B. Results

Fig. 3 shows the *actual* metric value and the one *predicted* by the model in Equation (1). Time series are shown by failure setting. Fig. 3 includes time series from the *Normal run* (Fig. 3a), i.e., no fault injected. Table I provides an overview of the detection outcome.

Active Hang. Fig. 3b shows that the predicted values closely mimic the actual values in fault-free records; on the top, the model can also predict abrupt increases on CPU usage. Nevertheless, we noted that in some records (*e.g.*, 92-94, 137-138), although the *presence* of spikes in CPU utilization was correctly predicted, the *magnitude* (or height) of the spikes was not predicted precisely. Around record 334 – when the active hang fault injection is started – it can be seen a sudden significant gap between the actual value of CPU the predicted value. Because of the gap, anomalies are detected with fairly good accuracy throughout the duration of the fault injection.

Passive Hang. Similar patterns can be observed in Fig. 3c where predicted values mimic actual values. Interestingly, there is no gap between predicted and actual value when injection is started around record 342. Having no TPs (meaning zero detection of anomalies) led us to an important observation. During a passive hang, the system goes into an indefinite waiting state for resources, causing a pause on operation activities; moreover, the passive hang did not affect CPU usage. This experiment highlights the existence of anomalies, which turn out to be **asymptomatic** in the relation of logs and metrics. We state that the detection technique assessed in this paper *is ineffective for those errors which simultaneously suppress the normative system activity and metric changes*. Asymptomatic anomalies can be reasonably addressed by *complementing* metrics-logs contextual detection with existing log-based failure analysis techniques.

Crash. Prediction fairly resembles the actual values before the injection; execution is aborted a few seconds after the injection around record 383 – see Fig. 3d. Noteworthy, the anomaly was detected as soon as the fault was injected, however the system crashed few seconds after the injection.

Although the approach detected all the positive records, precision appeared to be low. This indicates the existence of too many false positives.

V. Improvements

A. Change Detection

Above-presented experiments highlight that the predicted CPU values mimic the pattern of actual values when the records are failure-free. However, in some cases, it did not correctly predict the magnitude of the spikes. For example, in Fig. 3c (*Passive Hang*) record 160 shows a peak in CPU usage in both predicted and actual values. However, the difference between these two values is fairly large: the actual value indicates 76.16% CPU usage while the predicted one is 40.14%; moreover, both values are far bigger than the mean value of CPU usage, that is, 2.05%. Such big differences cause the detector to raise false anomalies.

This observation inspired us to investigate whether we can improve the accuracy of the anomaly detection approach from the perspective of **change detection** along with the prediction from the assertion equation. We then used a new threshold policy for detecting anomalies. This checks whether both predicted and actual values indicate a significant change from the mean. The proposed policy is described in Algorithm 1, where the actual value is denoted by ac, the predicted value by

Algorithm 1 - Change Detection

- Anomaly Detection:
1: **if** $(|ac - pr| < \epsilon)$ **then**
2: $anomaly \leftarrow false$
3: **else if** $(|ac - m| < \sigma$ AND $|pr - m| < \sigma)$ OR $(ac > m + \sigma$ AND $pr > m + \sigma)$ OR $(ac < m - \sigma$ AND $pr < m - \sigma)$ **then**
4: $anomaly \leftarrow false$
5: **else**
6: $anomaly \leftarrow true$
7: **end if**

TABLE II: Anomaly detection results (Change Detection)

	Active Hang	Passive Hang	Crash
Total Records	513	519	387
True Negatives	447	451	377
False Negatives	0	60	0
True Positives	62	0	4
False Positives	4	8	6
Precision	0.939	0.000	0.400
Recall	1.000	0.000	1.000
Accuracy	0.992	0.869	0.984

pr, standard deviation by σ, and standard error of estimate by ϵ. Table II shows the results obtained with the change detection policy. It can be noted that there was a reduction of the false positives for all three experiments with respect to the ones obtained *without* change detection (Table I).

B. Adaption of Time Windows

In our system, we note potential delay between an operation action and its effect(s) becoming observable, which may not be reflected into 1-second time windows. We investigated if a further improvement in terms of accuracy can be obtained by enlarging the observation period. We expanded the time window to a total of 3 seconds ($\pm$1 second from current time window) and re-ran our anomaly detection analysis. Table III shows the results obtained for both the original technique, i.e., *no CD*, and the one with the change detection algorithm, i.e., *w/ CD*. It can be noted that false positives reduced in all three failure settings for both *no CD* and *w/ CD* cases with respect to the ones obtained with the default time window (Table I and Table II). Noteworthy, a further increase of the time window did not lead to improvements of the results.

The obtained results suggest the larger time-window as another enhancement for the original technique. In addition, the results in Table III highlight again that the use of the change detection algorithm is beneficial in terms of false positives reduction.

VI. Conclusion

In this paper we analyzed the effectiveness of an anomaly detection approach based on log-metrics correlation from the literature [6] for an air traffic control system. Our initial evaluation revealed several weaknesses of using the approach in our setting, specifically the inability to detect asymptomatic anomalies like passive hangs, imprecision in predicting the magnitude of spikes, and overly localized view of 1*sec* time windows. We addressed these with suggested improvements, specifically change detection and 3*sec* time windows.The final evaluation detects anomalies with high accuracy and low delay. In the future, we plan to experiment with more types of anomalies to have better assessment of applicability of the approach. Many additional details can be found in our technical report [8].

TABLE III: Anomaly detection results (3-*seconds* Time Window; with (w/) or without (no) Change Detection (CD))

	Active Hang		Passive Hang		Crash	
Total Records	513		519		387	
	no CD	w/ CD	no CD	w/ CD	no CD	w/ CD
True Negatives	446	451	449	459	375	382
False Negatives	0	0	61	60	0	0
True Positives	62	62	0	0	4	4
False Positives	5	0	9	0	8	1
Precision	0.925	1.000	0.000	0.000	0.333	0.800
Recall	1.000	1.000	0.000	0.000	1.000	1.000
Accuracy	0.990	1.000	0.865	0.884	0.979	0.997

Acknowledgment

This research has been partially supported by the MINI-MINDS PON Project (n. B21C12000710005) funded by the Italian Ministry of Education, University and Research, and by Programme STAR, financially supported by UniNA and Compagnia di San Paolo under Project "Towards Cognitive Security Information and Event Management" (COSIEM).

References

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, pp. 15:1–15:54, 2009.
[2] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni, "Anomaly? Application change? Or workload change? Towards automated detection of application performance anomaly and change," in *Proc. Intl. Conf. on Dependable Systems and Networks (DSN)*, Jun. 2008, pp. 452–461.
[3] T. Wang, J. Wei, W. Zhang, H. Zhong, and T. Huang, "Workload-aware anomaly detection for web applications," *Journal of Systems and Software*, vol. 89, pp. 19–32, Mar. 2014.
[4] O. Ibidunmoye, F. Hernandez-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, pp. 1–35, 2015.
[5] N. Gurumdimma, A. Jhumka, M. Liakata, E. Chuah, and J. Browne, "CRUDE: combining resource usage data and error logs for accurate error detection in large-scale distributed systems," in *Proc. IEEE Symposium on Reliable Distributed Systems (SRDS)*, Sep. 2016, pp. 51–60.
[6] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, "Metric selection and anomaly detection for cloud operations using log and metric correlation analysis," *Journal of Systems and Software*, 2017.
[7] ——, "Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis," in *Proc. IEEE Intl. Symp. on Software Reliability Engineering (ISSRE)*, Nov. 2015, pp. 24–34.
[8] M. Farshchi, I. Weber, R. Della Corte, A. Pecchia, M. Cinque, Jean-Guy, and J. Grundy, "Technical report: Anomaly detection for a critical industrial system using context, logs and metrics," Universit di Napoli Federico II, Tech. Rep., 2018. [Online]. Available: http://www.fedoa.unina.it/id/eprint/11969
[9] M. Cinque, D. Cotroneo, R. Della Corte, and A. Pecchia, "Characterizing direct monitoring techniques in software systems," *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1665–1681, Dec. 2016.
[10] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004.