Technical Report: Anomaly Detection for a Critical Industrial System using Context, 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 and malfunctions at runtime. These techniques are highly relevant for critical software systems, where monitoring is often mandated by international standards and guidelines. In this technical report, 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. We propose an automated abstraction approach to infer system activities from dense logs and use regression analysis to infer the anomaly detector. We observed that the detection accuracy is impacted by abrupt changes in resource metrics or when anomalies are asymptomatic in both resource metrics and event logs. 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.**

This technical report accompanies the paper "Contextual Anomaly Detection for a Critical Industrial System based on Logs and Metrics" [1] and provides further details on the analysis method, case study and experimental results.

***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) [2], [3]. 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. For example, resource utilization has been supplemented with volume and latency of transactions [4], and response time in virtualized infrastructures [5] and web-based applications [6], for better anomaly detection. Recent work attempts to combine resource metrics and *event logs* [7], [8] for contextual anomaly detection. This is a promising direction because **event logs** are ubiquitously emitted by computer systems and contain large amounts of data on regular and erroneous events occurring during operation [9], [10]. Logs have in fact been successfully used for failure analysis over the past decades. However, the approach of combining metrics and event logs poses several major challenges. Most notably, logs consist of unstructured text, which underlies the need for transformation techniques before integration with numerical data sources (*e.g.*, resource utilization, response time). Further challenges include the inference of statistically meaningful relationships between metrics and logs, determining the most sensitive metrics, and building a model to define assertions for contextual anomaly detection.

This technical report explores the use of such contextual anomaly detection in a critical industrial system. The aim of this report is twofold: (i) experimenting with a metrics-logs contextual anomaly detection method in a new compelling domain, to gain data-driven feedback for potential improvements to the method, and (ii) contributing real-world case study experience for this technique.

Context-based techniques leveraging metrics-logs have been previously proposed to assess HPC jobs through *TACC_stats* resource utilization data [7] and cloud rolling upgrade operations based on *CloudWatch* [8]. We apply the approach to industrial middleware supporting the integration of time-sensitive Air Traffic Control (ATC) applications[1] based on our previous work in [8]. To the best of our knowledge, the combination of metrics-logs has been not previously been explored for mission critical systems. We start from the contextual anomaly detection technique proposed in [8], which is non-intrusive and unsupervised. The technique is potentially valuable for critical software systems, such as the one addressed by this study, where monitoring is recommended – if not mandatory – by international standards and guidelines (*e.g.*, IEC 61508-7, ISO-26262, DoD RAM Guide). Metrics and logs are a *byproduct* of such a system execution that can be leveraged for anomaly detection. Different from previous work, the target system here generates dense and large volume of logs (containing up to 240 lines per second, and 155 distinct event types) and resource utilization metrics have a fine-grained 100-millisecond sampling period. We propose a method to infer high-level system activities from the raw log data. Regression analysis is then used to assess the correlation between metrics and system activities (automatically inferred

[1] Middleware and applications are developed by a world-leading company in electronic and information technologies for defence, aerospace, and land security. They have been made available within an industry-academia collaboration project. Company and project are not disclosed here for double-blind reviewing reasons.

from log data) and to derive the anomaly detector. We emulated regular and failure executions of the ATC system by running it in a controlled testing environment; around 2000 observations of metrics/context have been collected during the experiments. We measured the effectiveness of the anomaly detection technique by using typical metrics, such as precision, recall, and accuracy. The key contributions of this work are:

- *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. The method combines (i) POD-Discovery [11] to generate regular expressions for each type and (ii) *interpolated occurrence strength* measurements to cluster event types into activities. Particularlly, this addresses the limitation of applying the previous method [8] on dense logs of common application operation environments. We infer 17 system activities out of the initial 155 types.
- *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 – although the regression technique correctly predicts spikes in resource utilization – a large difference between the predicted and actual resource utilization causes sporadic false positives. Moreover, we found that contextual anomaly detection is ineffective for errors suppressing both the normative system activity and metric changes.
- *Means for improving contextual anomaly detection*. Based on the results of our experimental analysis, we proposed, implemented, and evaluated means for improving anomaly detection. A *change detection* algorithm is proposed for significantly reducing the number of false positives. We also propose a time-window-based approach, which enlarges the observation period and further increases detection accuracy.

The rest of the technical report is organized as follows: in Section II, we discuss relevant approaches. Section III provides an overview of the selected analysis method, followed by sections IV, V, and VI presenting the reference ATC system and how the method was tailored for it. Section VII presents the assessment of contextual anomaly detection, while Section VIII describes and evaluates the proposed improvements. Sections IX discusses threats to validity. The report concludes with a summary of the key contribution in Section X.

II. Relevant and Related Work

Anomaly detection has been broadly employed for system health and performance monitoring. The majority of the anomaly detection approaches have focused on point-based techniques [3]. These techniques do not take into account the impact of the dynamic nature of workload or the legitimate contextual and behavioral factors that cause anomalous spikes on system resource utilization. This issue motivated several studies to propose context-based techniques, considering a set of conditions or behavioral attributes of a system.

Wang *et al.* propose an approach based on workload profiling using an incremental k-means clustering technique to recognize access patterns and request volume from the workload [12]. Local outlier factor (LOF), a machine learning technique which works based on a concept of a local density, was then employed to identify anomalous data instances for each type of workload pattern. In this method, by measuring the local density of a data point in comparison to the local densities of its neighbors, the areas of similar density can be detected, and objects that have a considerably smaller density than their neighbors are tagged as anomalies. The above approach has the advantage of being independent of domain knowledge. However, it is not suitable for fine-grained monitoring due to many clusters generated from workload patterns [13]. It also has high computational complexity due to workload pattern recognition and LOF calculation for each arrival data instance. In addition, it is a supervised approach, and hence requires labeled data.

In another study, Cherkasova *et al.* [4] present a regression-based approach to model the resource consumption of Web applications. They present a profiling method by identifying application performance signature (using transaction count, transaction latency, count of database calls, and latency of outbound calls) to model the run-time application behavior, but do not use log events. In addition, the exploratory/causation aspect of the regression model is not used, in contrast to the approach we used here.

Magalhaes and Silva [6] introduce an approach to detect root-cause factors of observed performance variations due to workload changes or application updates. This is done by adopting the Pearson coefficient of correlation between system metrics and aggregated workload. They employed Aspect Oriented Programming (AOP) to monitor the response time of every transaction and then Person correlation to model their correlation. This approach is limited to just one metric of workload at a time [14]. Moreover, by changing the source code their approach is intrusive.

Pecchia *et al.* [15] assess the use of *likely* system invariants –i.e., properties of a program or a system expected or observed to hold in normative executions– for anomaly detection. Invariants are inferred from monitoring logs in systems performing batch work; invariants are based on typical metrics, such as job/task completion time, CPU usage and status codes. The key aspect of their proposal is abstracting workloads by means of attributes, and inferring invariant relationships among attributes. Evaluation is done with real datasets from a Google cluster, whose traces are publicly available, and a Software-as-a-Service platform.

Kang *et al.* [5] proposed DAPA (Diagnosing Application Performance Anomalies), a statistical approach to model the quantitative relationship between the application response time and virtualized system metrics based on SLA violations. The main criterion in their study for monitoring anomalies where the indicators of SLA violation. However, anomaly detection target metrics are not system resource metrics and application response time is the only source of monitoring information for detecting SLA violation. Being dependent on a single metric, this approach is limited to the detection of failures that lead to immediate performance anomalies.

Gurumdimma *et al.* [7] take advantage of both application logs and resource metrics and propose a method called CRUDE (Combining Resource Usage Data and Error Logs) to detect errors. This relies on the computation of mutual information, entropy and anomaly score, to identify the chain of events that may lead to a failure. Logs are analyzed with hierarchical clustering and feature extraction methods, and Principal Component Analysis (PCA) is employed to detect anomalous jobs within a period. CRUDE assumes that a higher entropy (uncertainty) with reduced mutual information denotes abnormal system behavior or a failure sequence, with the opposite signifying normal behavior. This approach detects sequence anomalies (rather than point-anomalies) based on the sequence of events in relatively long time-windows (*e.g.*, 60 minutes). In contrast, we aim to detect point anomalies in a relatively small time window (within seconds).

The most suitable approach for our case study is arguably the one by Farshchi *et al.* [8], [16]. It is non-intrusive and unsupervised, that is, requires neither changes to the target system nor labeled training data. Farshchi *et al.* [8], [16] use statistical methods to select metrics and cluster event types, and specifically regression analysis to find correlations between relevant log events and metric changes. Like Gurumdimma *et al.* [7], this approach can contextualize metric changes through log events; but it can do so for smaller time windows (few minutes in [8], [16]), and work point-based (needing fewer observations). In the approach by Gurumdimma *et al.* [7], setting thresholds for various metrics needs repeated observations and readjustment to find an optimal result. Instead, [8], [16] can set thresholds based on the error estimates of the regression model (although it remains configurable and can be overridden if needed). *We therefore decided to use the most recent and complete version [8] of this approach in our analysis.*

The domain of anomaly detection is large, and there are several systematic surveys [2], [3], [17], [18]. A highlighted issue in the domain of system monitoring is the lack of cross-layer monitoring [19]. This is a challenging task, as it is difficult to map different monitoring data types and to interpret them in an integrated form. Our study contributes in this direction as it considers two different types of monitoring information (*i.e.* event logs and resource metrics) which can span multiple layers.

III. Overview of Analysis Method

The contextual anomaly detection technique proposed in [8] 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 model is then used to generate assertions to be used for anomaly detection.

The technique requires (i) a stream of time-stamped events, such as events represented 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. As illustrated in Fig. 1, events generally happen at *irregular* time intervals whereas resource metrics are collected at regular intervals. In the following, we briefly describe the main steps of the base technique (for details, please refer to [8]).

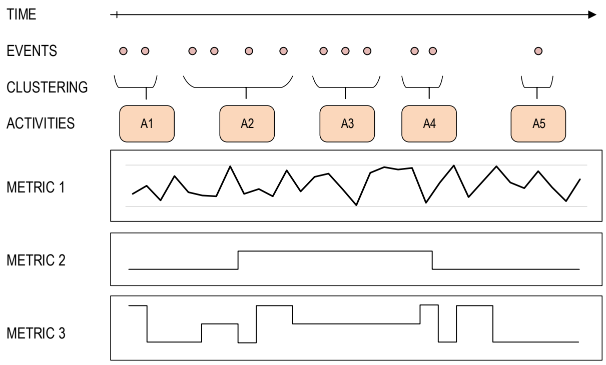


Fig. 1: The occurrence of log events and metric data over time – A1..A5 denote activities; Metric1..3 denote metrics such as CPU usage, RAM usage, heap size etc. Adapted from [8].

- *Data Collection and Preparation:* set up and running of a suitable experiment, collection of metric data and log files, and translating metric information into a format suitable for further processing (if required).
- *Log Event Type Extraction:* clustering of (generally low-level) event traces, identification of unique *event types*, and generation of regular expressions for each identified event type.
- *Representing Log Event Type as Quantitative Metric:* identification of a suitable anomaly detection time window and extraction of interpolated occurrence of event types for each consecutive time window.
- *Log Event Type Correlation Clustering into Activities:* clustering of highly correlated event types and identification of high-level *activities*.
- *Metric Filtering and Aggregation:* given the collected raw metrics, initial selection of suitable metrics for further processing and, if required, definition of new, case-study specific aggregated and/or derived metrics.
- *Event-Metric Correlation Derivation:* using a suitable regression model, derive an events-metric correlation model (with event type occurrences as independent variables and a metric as dependent variable) for each of the metrics chosen in the previous step.
- *Target Metric selection:* identify 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.
- *Assertion specification for anomaly detection:* generation of an assertion specification for the model(s) selected in the previous step.

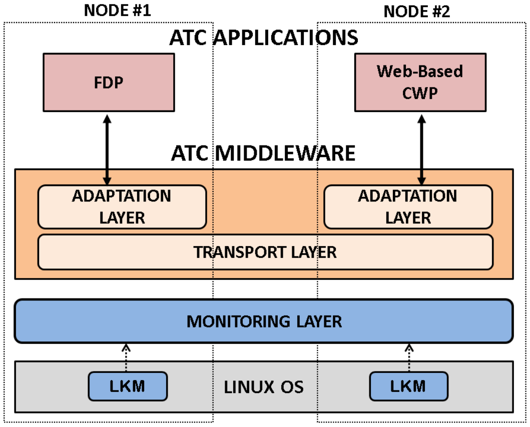


Fig. 2: High-level architecture of the reference system

IV. Case Study System

System overview. The example system considered in this study is a middleware platform for the integration of mission-critical applications in the Air Traffic Control (ATC) domain.

The high-level architecture of the system is shown in Fig. 2. It consists of the middleware and two ATC applications: (i) a Flight Data Processor (FDP), which generates/updates flight data (*i.e.* data describing a flight, such as arrival/departure time and flight trajectory) and publishes the data on the middleware, 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, implemented on top of the JBoss application server (http://jbossas.jboss.org/). The transport layer ensures the communication between the FDP and CWP, according to the publish-subscribe paradigm and using a commercial implementation of the OMG Data Distribution System (http://www.omg.org/dds/). The adaptation layers allow applications to use the middleware and its services. The monitoring service uses a Linux loadable kernel module (LKM) to collect data through system probes. LKM allows accessing metrics about each running process stored by the Linux kernel in process descriptors, such as the open files of the process, its state and resources usage (*e.g.*, CPU and RAM).

Testbed. The middleware has been deployed in a configuration consisting of two nodes (Intel Xeon E5-1620 v2 with 8 cores, 16 GB of RAM, 1 GB/s network interface and Ubuntu 14.04) that emulate the ATC system, as illustrated in Fig. 2. The two nodes run the FDP application and the web-based CWP, respectively, on the middleware.

The OS processes running the FDP and the middleware are monitored by the kernel probes with a 100-millisecond sampling rate. It should be noted that – although we deployed a controlled testbed for experimental purposes – both the ATC middleware and applications consist of the real-world software made available by an industrial partner for research purposes.

Workload. The system is exercised by a test suite used by the industry provider to emulate real usage during operations. The test suite generates and updates flight information, trough the FDP. Data are published through the middleware and consumed by the CWP, presenting them on a web console.

V. Log Analysis

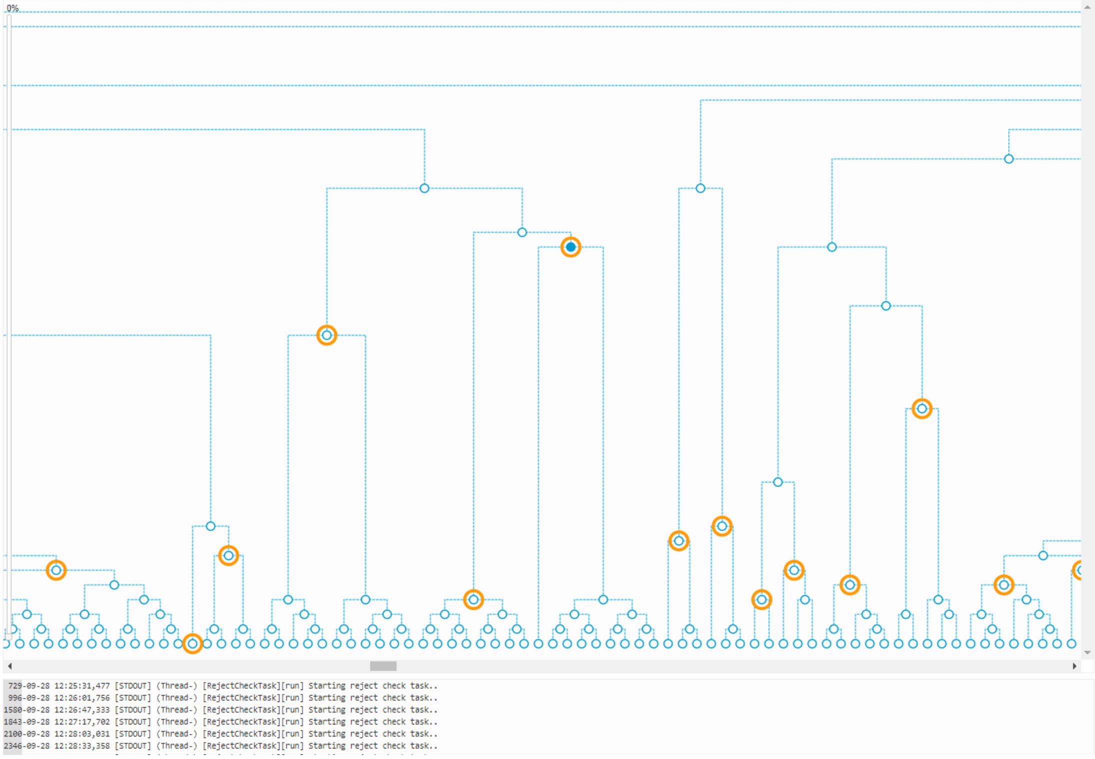
This section describes how we process log files, identify unique log event types, represent log events in a quantitative form and cluster highly correlated log event types into a set of log activities. The prerequisite step to process logs in the considered approach is to make sure that log events include timestamps. Timestamps are necessary to track log events and also to map the occurrence of log events to the metric observations.

A. Log Event Type Extraction

To observe how the activities reported in log events change the state of resources, we are interested in tracking the occurrence of log events. We needed a way to parse and trace log events, and for this purpose we employ regular expressions to derive a template of event logs. The goal of this step is to extract the pattern of recurring event logs by automatically separating the constant parts and variable parts of a raw log message, and further derive a regular expression to associate each log message with a specific *log event type*.

Log files are processed as follows: first, the timestamps and log description are extracted for each log line, followed by tokenization of the log message. Next, regular expressions are generated for each token of the log message. Then, message tokens are divided into two parts: constant tokens and variable tokens, by analyzing the patterns of regular expressions using a set of pre-defined rules. Lastly, the generated regular expressions are combined to represent a unique log event type. For each new log line, the log event is compared by pattern matching with regular expressions, and if a pattern is not found, the above steps are repeated for the new log line.

The middleware generates dense and large volume of logs. For example, it can generate around 2,700 lines during five minutes of operations and up to 240 lines per second. Unlike the way performed in [8], extracting regular expressions and finding unique event types in such a volume of log lines cannot be done in a manual way. Therefore, we employed a log abstraction tool, POD-Discovery [11], to generate regular expressions for unique log events. POD-Discovery is used to cluster low-level event traces into higher-level events. POD-Discovery tokenizes each log message into several tokens. The tokens of a log message can be divided into two parts: the constant parts and the variable parts [20]. The constant tokens of a recurring log event remain the same for a recurring log event by default, while the variable tokens hold the runtime information of a recurring log event, for example an IP address or a port number. POD-Discovery employs a token distance measure using the Levenshtein distance [21] for string comparison. This method is used as a metric to know how many similarities exist between a token of one log event and another one. For pattern extraction of unique log events from logs in our study, we began to set a distance measure of



```
 729-09-28 12:25:31,477 [STDOUT] (Thread-) [RejectCheckTask][run] Starting reject check task..
 996-09-28 12:26:01,756 [STDOUT] (Thread-) [RejectCheckTask][run] Starting reject check task..
1580-09-28 12:26:47,333 [STDOUT] (Thread-) [RejectCheckTask][run] Starting reject check task..
1843-09-28 12:27:17,702 [STDOUT] (Thread-) [RejectCheckTask][run] Starting reject check task..
2100-09-28 12:28:03,031 [STDOUT] (Thread-) [RejectCheckTask][run] Starting reject check task..
2346-09-28 12:28:33,358 [STDOUT] (Thread-) [RejectCheckTask][run] Starting reject check task..
```

Fig. 3: A screenshot of POD-Discovery - yellow circles show the tree nodes at 10% similarities, the selected node shows the log events under the tree hierarchy of that node at the bottom of the screen

minimum similarities and generated log event types. Our initial inspection of event types at the minimum level showed that they were too low, as there were similar log events dispersed over two or three event types. After trying this and a few rounds of inspection and gradually increasing similarity levels, we found that a 20% distance threshold gives us a level which grouped similar log events into individual log event types. The output of processing the log of the middleware led to 155 unique log event types.

B. Representing Log Event Type as Quantitative Metric

Middleware logs in the case study are available with the precision of milliseconds. However, the occurrence of logs varies between a few milliseconds to a few seconds. Monitoring metrics are reported in approx. 100-millisecond (ms) intervals.Given the interval occurrence of logs and the availability of the data, we decided to choose a one-second interval as our default time window. This time window is small enough to provide fine-grained monitoring but not so small that the lasting impact of the operation's action on resources would lead to too many false alarms.

We use an approach that converts the occurrence of log events based on the interpolated occurrence strength of each event type, as illustrated in the following. Given a log event type, denoted as e, and the smallest unit of time that logs can track, denoted as x, the current time window, denoted as tw, and D_{tw} represents the duration of the time window, then the weight-timing occurrence of an event type at time x of a current time window and the next time window can be obtained:

$$e_{n(tw)} = \frac{D_{tw} - x}{D_{tw}} \qquad e_{n(tw+1)} = \frac{x}{D_{tw}} \tag{1}$$

For the sum of n occurrences of an event type in a time window, we have:

$$E_{tw} = \sum_{i=1}^{i=n} e_{n(tw)} \tag{2}$$

For example, consider the two log events:

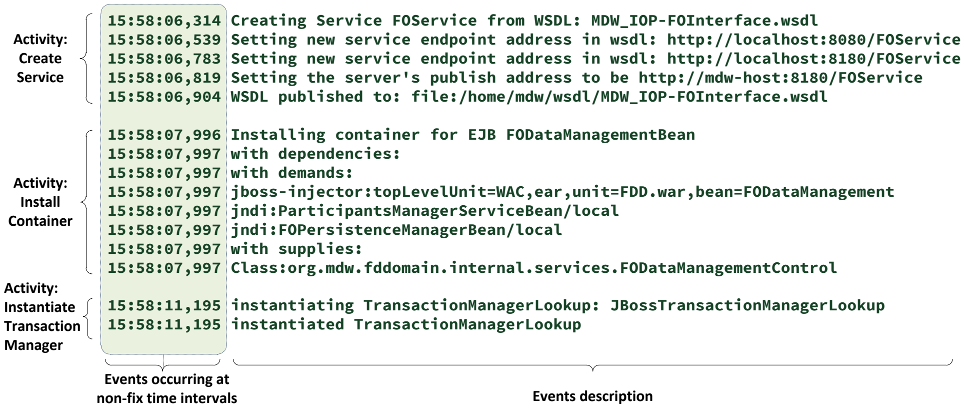
```
2015-09-28 12:26:16,562 INFO [MDW.FDD] (Thread-37) [
        FDD_FlightDataListener_Impl] Class Loader Updated

2015-09-28 12:26:16,974 INFO [MDW.FDD] (Thread-37) [
        FDD_FlightDataListener_Impl] Class Loader Updated
```

These log events are of the same event type and occurred twice within a one-second time window with the timestamp *12:26:16*; at 562 and 974ms, denoted as e_1 and e'_1, respectively. The interpolated occurrence strength is as follows:

$$e1_{12:26:16} = \frac{1000 - 562}{1000} = 0.438 \qquad e1_{12:26:17} = \frac{562}{1000} = 0.562$$

$$e1'_{12:26:16} = \frac{1000 - 974}{1000} = 0.026 \qquad e1'_{12:26:17} = \frac{974}{1000} = 0.974$$



```
15:58:06,314 Creating Service FOService from WSDL: MDW_IOP-FOInterface.wsdl
15:58:06,539 Setting new service endpoint address in wsdl: http://localhost:8080/FOService
15:58:06,783 Setting new service endpoint address in wsdl: http://localhost:8180/FOService
15:58:06,819 Setting the server's publish address to be http://mdw-host:8180/FOService
15:58:06,904 WSDL published to: file:/home/mdw/wsdl/MDW_IOP-FOInterface.wsdl

15:58:07,996 Installing container for EJB FODataManagementBean
15:58:07,997 with dependencies:
15:58:07,997 with demands:
15:58:07,997 jboss-injector:topLevelUnit=WAC,ear,unit=FDD.war,bean=FODataManagement
15:58:07,997 jndi:ParticipantsManagerServiceBean/local
15:58:07,997 jndi:FOPersistenceManagerBean/local
15:58:07,997 with supplies:
15:58:07,997 Class:org.mdw.fddomain.internal.services.FODataManagementControl

15:58:11,195 instantiating TransactionManagerLookup: JBossTransactionManagerLookup
15:58:11,195 instantiated TransactionManagerLookup
```

Activity: Create Service
Activity: Install Container
Activity: Instantiate Transaction Manager
Events occurring at non-fix time intervals
Events description

Fig. 4: An example of JBoss logs with clustering related log events to set of activities (original log messages are shortened to improve the presentation).

Then the interpolated occurrence strength for $e1$ in the current and next time window are determined as follows:

$$E1_{12:26:16} = 0.438 + 0.026 = 0.464$$

$$E1_{12:26:17} = 0.562 + 0.974 = 1.536$$

As the result shows, the impact of the occurrence of log events is distributed between two time windows (two seconds), where we have a higher interpolated occurrence strength for the second time window compared to the first time window. The interpolated occurrence strength is determined in the same way for all other event types across all observed time windows.

C. Log Event Type Correlation Clustering

Systems application and operation logs are often at a more fine-grained level than the system activities that affect the status of resources. Most often a system behavior is not characterized by a single log event – typically a set of log events together cause a tangible impact on the status of resources. Therefore, to find the impact of event logs on resources we cluster related log events.

The aim of event type clustering is to identify those event types with highly correlated occurrences. For this purpose, we use the Pearson correlation coefficient, commonly represented by the symbol r. Given two datasets $x_1, ..., x_n$ and $y_1, ..., y_n$ containing n values each, r is defined as follows:

$$r = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{[n\sum x_i^2 - (\sum x_i)^2][n\sum y_i^2 - (\sum y_i)^2]}} \tag{3}$$

In calculating the correlation occurrence between two log event types:

- n is the number of monitoring observations.
- x_i denotes the interpolated occurrence strength of event type x at time i, e.g., $E1_{12:26:16}$ above.
- y_i denotes the interpolated occurrence strength of event type y at time i, e.g., $E2_{12:26:16}$.

The Pearson correlation coefficient r ranges from -1 to $+1$. A value of zero or very close to zero indicates that there is no occurrence correlation between two log event types. A value close to 1 indicates a strong positive correlation whereas a negative correlation coefficient indicates that two event types rarely co-occur. We defined a rule that, for event types to be grouped into the same cluster, they must have a correlation strength of more than 75% ($r > 0.75$).

As a result of this correlation analysis, the event types of the middleware logs for our ATC applications are grouped into 17 log activities. Fig. 4 shows a sample of log events that belong to separate activities in the application server. By looking at the timestamps, we can observe that the log events are happening with different time intervals, with some sets of log events occurring with closer timestamp than the others. The number of log event types associated with each activity varies widely from one event type to over 20 event types in a group. We labeled these $A01$ to $A17$.

Note that this process of log abstraction, unlike many log abstraction techniques that analyze the context of logs by using pattern signature and feature extraction methods [22], [23], [24], [25], [26], relies solely on statistical analysis based on interpolated occurrence strength. No domain knowledge is used or required for this process.

VI. Model Building

A. Metric Selection

In this section, we describe how we identify metrics that have the highest sensitivity to the log activities from the operation. For our ATC case study, we analyzed middleware monitoring data collected from system probes by the Loadable Kernel Module. The monitoring data includes several non-application[2] metrics with timestamps of approximately 100-millisecond frequency.

It is worth noting that there were some other monitoring metrics included in the data of the case study such as Load Average and Network traffic-related metrics show sensitivity

[2] Application metrics with valid timestamps were not available in our data set. Therefore, from all the available metrics those ones that had valid timestamp were used in our analysis.

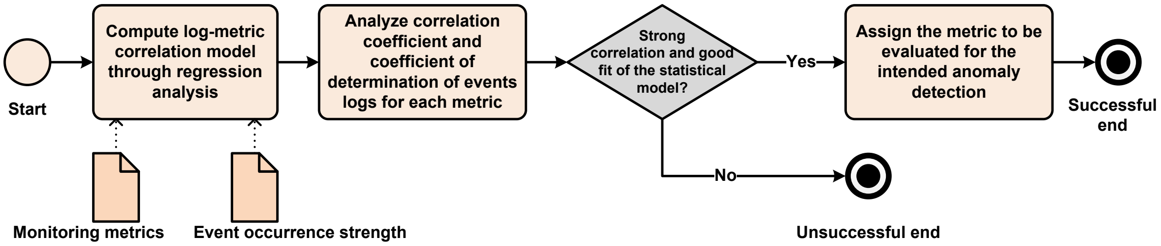

Start
Compute log-metric correlation model through regression analysis
Analyze correlation coefficient and coefficient of determination of events logs for each metric
Strong correlation and good fit of the statistical model?
Yes
Assign the metric to be evaluated for the intended anomaly detection
Successful end
No
Unsuccessful end
Monitoring metrics
Event occurrence strength

Fig. 5: Checking the relevance of a monitoring metric. Adapted from [8].

TABLE I: List of available metrics.

Metric	Validity Status
CPU usage	Valid
number of voluntary context switches	Not Valid
number of involuntary context switches	Not Valid
RAM usage	Valid
VM current size	Valid
VM peak size	Valid
VM currently resident in RAM	Valid
peak of VM resident in RAM	Valid
VM size for data	Valid
VM size for stack	Not Valid
VM size for code	Not Valid
number of page faults	Valid
disk read	Not Valid
disk write	Not Valid
number of opened files	Valid
number of sockets	Valid
heap size	Valid

to the activities of operation. However, due to the lack of timestamps for monitoring records, we had to disregard them for the analysis. This is because timestamp is a pre-requisite for our analysis in order to map the log activities to the monitoring metric data. Table I shows the list of metrics of monitoring data that included timestamps. As can be seen in Table I, Some of the metrics that were available could not be used for statistical analysis as they did not come with complete data: their values were filled with zero or just one constant value was recorded for all records. Therefore, we dropped these metrics from our analysis; the valid metrics are listed in Table II. As a result of this filtering, we ended with 11 metrics that were suitable for the statistical analysis. We used 11 metrics for the analysis, listed in Table II.

We followed the process given in Fig. 5 and used the monitoring metrics, along with the metrics derived from the log activities, and performed regression analysis to assess the sensitivity of each target monitoring metric to the system operational activities reported in the logs.

Multiple regression is done for several independent variables (IV) as predictors (*i.e.*, activities clustered from event logs), and one dependent variable (DV) (*i.e.*, a monitoring metric) as the outcome. An objective of applying regression technique is to derive a model from input sample data with the minimum absolute (squared) error [27]. Given:

- y is the dependent variable.
- $\beta_1, \beta_2, \ldots, \beta_p$ are the regression parameters.
- $x_1, x_2, \ldots, x_p$ are the independent variables, also called predictors. These are the interpolated occurrence strengths as per Equation 2 for a time window, for the event types representing a given activity.
- ϵ is the error of estimate. The residual $\epsilon_i = y_i - \overline{y_i}$ is the difference between the value of the dependent variable predicted by the model, $\overline{y_i}$, and the true value of the dependent variable, y_i.
- α denotes a constant value as an intercept, where it indicates the mean value of y when all $x\!=\!0$.

The general form of a multiple linear regression function is

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon \tag{4}$$

The coefficients $\beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$ denote the effect of each variable on an overall model. The coefficient parameters measure the individual contribution of independent variables to the prediction of the dependent variable, after taking into account the effect of all the independent variables [27].

Once a multiple regression equation has been constructed, we can check how strong the regression output is in terms of (i) correlation of the event logs with the target metrics, and (ii) the model's predictive abilities.

Given a sufficiently large number n of records of data of a sample population, independent variables as predictors, denoted by $x_1 \ldots x_p$ (log activities), the *observed* dependent variable, denoted by y_i (actual value of metric), the *estimation* of y_i using the regression model (also called the *fitted response*), denoted by f_i (predicted value of the metric), with $\bar{y}$ being the mean of the observed dependent variable and $\overline{f_i}$ the mean of the estimations is:

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{5}$$

The total sum of squares is obtained as:

$$SS_{tot} = \sum_{i=1}^{n} (y_i - \bar{y})^2 \tag{6}$$

Similarly, the sum of squares of residuals is defined as:

$$SS_{res} = \sum_{i=1}^{n} (y_i - f_i)^2 = \sum_{i=1}^{n} \epsilon_i^2 \tag{7}$$

TABLE II: Coefficient correlation and coefficient determination results for each valid metric

Metric	R	R^2	$Adj.R^2$	*p-value*
CPU Usage	0.877	0.769	0.761	0.000
VM current size	0.716	0.513	0.498	0.000
VM peak size	0.542	0.294	0.271	0.000
Number of sockets	0.381	0.145	0.116	0.000
Number of opened files	0.380	0.145	0.115	0.000
VM size for data	0.308	0.095	0.064	0.000
Number of page faults	0.273	0.074	0.045	0.000
Heap size	0.246	0.060	0.028	0.021
VM in RAM(peak)	0.208	0.043	0.012	0.141
RAM usage	0.198	0.039	0.008	0.218
VM in RAM(current)	0.186	0.034	0.003	0.000

Using equations 6 and 7, we can determine how much variation of a dependent variable can be explained by a predictor. The coefficient of determination R^2 is defined as [27]:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{8}$$

R^2 indicates how well a model predicts new observations, and can be used to assess the predictive power of a regression model for the given predictors and target variables [27], respectively. $Adj.R^2$ is a slightly more conservative version of R^2 that penalizes a high number of predictor variables in a model [27]. $Adj.R^2$ is always equal or less than R^2 and the difference between R^2 and $Adj.R^2$ gets smaller as the sample size increases. With p being the total number of independent variables in the model and n is the sample size, $Adj.R^2$ is defined as:

$$Adj.R^2 = 1 - \frac{n-1}{n-p}(1 - R^2) \tag{9}$$

In order to identify the most relevant metrics, the metrics from the log activities are taken as the predictor variables in the regression analysis, and each resource metric is taken as a target metric. The result of applying regression analysis is shown in Table II. R denotes the correlation between a given monitoring metric and the occurrences of activities from the event logs, and *p-value* the corresponding statistical significance level. Table II indicates that the only metric with a high prediction ability is *CPU usage*: $R^2 = 0.796$ and $Adj.R^2 = 0.761$, respectively. The next "best" metric is *VM current size*, with $R^2 = 0.513$ and $Adj.R^2 = 0.498$, indicating a far weaker prediction ability compared to CPU usage. The remaining metrics show fairly low to almost no correlation to the log activities.

From this analysis, CPU usage emerges as the best candidate metric for contextual anomaly detection for our ATC system. Its strong correlation value suggests that the variation in CPU usage should be explained by our regression model; conversely, changes in CPU usage values that are not be predicted from the log activities may be a good indicator of anomalies. We investigate this hypothesis in the next section.

TABLE III: Coefficients for identified influential factors

Predictors	β	Std. Error	B	*p-value*
Intercept	1.480	0.104	—	0.000
A01	0.186	0.770	0.008	0.810
A02	-0.363	0.818	0.010	0.658
A03	-0.257	1.075	0.057	0.243
A04	-0.135	1.180	0.006	0.909
A05	4.975	2.231	0.085	0.260
A06	3.981	1.863	0.066	0.033
A07	19.799	1.091	0.405	0.000
A08	3.766	0.208	0.403	0.000
A09	-3.108	2.791	0.042	0.266
A10	-38.030	6.065	0.352	0.000
A11	4.368	1.970	0.086	0.027
A12	0.719	0.406	0.044	0.047
A13	11.157	2.348	0.106	0.012
A14	-14.192	5.904	0.145	0.001
A15	7.723	2.356	0.073	0.000
A16	48.403	5.112	0.000	0.000
A17	3.901	0.515	0.816	0.000

*Note. β = Unstandardized regression coefficient;
B = Standardized regression coefficient.

B. Assertion Derivation

One of the objectives of performing multiple regression analysis is to find an explanatory relationship between the independent variables (activities extracted from logs) and the dependent variables (monitoring metrics). In the previous section, we identified CPU usage as a good candidate metric as it had high correlation with the log activities. In this section, we analyze which of the log activities are affecting the target metric (CPU usage), in order to derive assertion specifications for the employed anomaly detection approach. To perform this analysis, we take the regression coefficient for each predictive variable of our multiple regression models generated by the regression analysis. In this process, we take log activities as input predictor variables and CPU usage as sole target variable for the regression analysis.

The results are shown in Table III. The key indicator for identifying predictors that do not have significance on a regression model are Standardized regression coefficient(B) and *p-value*. Therefore, by checking these two factors, we observe that the coefficients of the activities A01, A02, A03, A04, A05 and A09 are statistically insignificant ($p > .05$).[3] These observations allowed us to narrow the set of contributing activities down to the 11 activities.

Rerunning multiple regression with the 11 activities that have statistical significance ($p \leq .05$) resulted in the outcomes shown in Table IV. By considering the standardized coefficient values in Table IV, we can assess the predictive power of each log activity for the CPU usage metric. In addition, we use the unstandardized coefficient (β) values from Table IV and regression function from Equation 4 to derive the assertion

[3] The p-value of 0.05 is commonly chosen as an acceptable level of significance [28], [29].

TABLE IV: Coefficient for identified influential factors after filtering

Predictors	β	Std. Error	B	*p-value*
Intercept	1.462	0.101	—	0.000
A06	4.606	1.819	0.077	0.012
A07	19.808	1.819	0.405	0.000
A08	3.766	0.208	0.403	0.000
A10	-30.411	3.955	0.281	0.000
A11	2.539	1.128	0.050	0.025
A12	0.797	0.394	0.049	0.025
A13	11.176	2.344	0.106	0.044
A14	-9.358	3.699	0.091	0.000
A15	7.741	2.351	0.073	0.012
A16	41.415	2.890	0.698	0.000
A17	3.557	0.463	0.216	0.000

*Note. β = Unstandardized regression coefficient;
B = Standardized regression coefficient.

equation (Equation 10) that can be used for predicting the CPU usage at each second.

$$y_i = 1.462 + 4.606 * A06_i + ... + 3.557 * A17_i \qquad (10)$$

At each time t_i, the actual value is expected to be predicted from the regression equation with an error of estimate of ± 6.059. Once the assertion is derived we can employ that for monitoring the metric values at run-time and detect likely anomalies when the prediction significantly deviates from the actual values. In our experiment, we consider the difference of the predicted and the actual value of CPU usage that is within the error of estimate to be normal, $|ac - pr| < \epsilon$, otherwise the corresponding records is registered as an anomalous record.

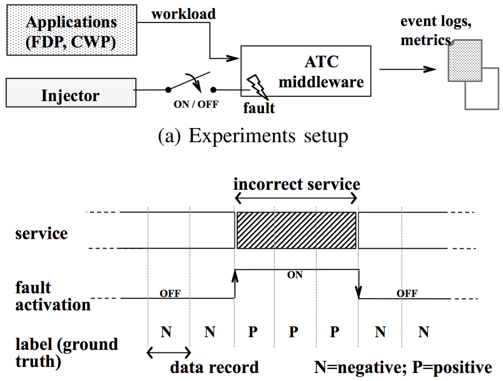
VII. Anomaly Detection with Initial Approach

The *model building* phase allowed us to identify the most sensitive metric, CPU utilization, to infer the activities in the log that most contribute to changes in this metric, and to derive an assertion equation that can be used to predict CPU usage from log event occurrence. We leverage this obtained model for our contextual anomaly detection approach. It is important to note that we performed our analysis based on four separate experiments. In the normal run (without any fault injection), we learned from normal run and derived the model. In this section, we validate the accuracy of the learned model on three new experiments. Collection of monitoring data, evaluation metrics and results are presented below.

A. Data collection and evaluation metrics

Regular and anomalous 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.



(a) Experiments setup

(b) Data record collection

Fig. 6: Fault injection setup and records collection

- **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 [30]. It should be noted that these failure settings are not meant to be exhaustive, but are sufficient to elicit a number of stressful conditions requiring improvements for contextual anomaly detection, which are discussed in Section VII-B.

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. The experiments have been conducted based on around 12000 log lines (after cleansing process) and around 18000 metric points.

Fig. 6a shows the experimental setup. Fig. 6b depicts the timing of the experiments. For each run, the ATC middleware and applications (FDP, CWP) are initialized and then the applications generate the workload described in Section IV. 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. 6b). 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 emitted by the approach contains the information of 1-second time windows, namely the actual value of CPU usage, the predicted value of CPU usage based on the assertion equation from log activities, the anomaly status of the record based on the comparison of actual and predicted metric, and the status of fault injection for the record. Given the above data, records are **labeled** as *positive* (P) if collected under incorrect service; *negative* (N) otherwise (Fig. 6b). We use the label as *ground truth*, or *oracle*, to assess the outcome of the anomaly detector as per the confusion matrix in Fig. 7. For example, a positive record (fault injection active) deemed

		label (ground truth)	
		negative	*positive*
detector	*negative*	true negative (TN)	false negative (FN)
	positive	false positive (FP)	true positive (TP)

Fig. 7: Confusion matrix

negative by the detector (no anomaly detected) represents a false negative (FN); similarly, a positive record flagged as positive by the detector is a true positive (TP). FN and FP mark cases where the detector did not work perfectly. The four categories in Fig. 7 are the basis for calculating precision, recall and accuracy values for the experiments [31].

B. Results

We ran anomaly detector against the labeled data records collected during the experiments. Fig. 8 shows the difference between *actual* metric value and the one *predicted* by the regression model. Time series are shown by failure setting. Fig. 8 also includes time series from a *Normal run* (Fig. 8a), where no fault has been injected. This is used as a reference of normal system behavior under the considered workload. Table V provides an overview of the detection outcome in terms of precision, recall, and accuracy. We discuss the anomaly detection results below.

Active Hang. Fig. 8b 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, 153-154, 488) the difference between actual-predicted values is significant, which causes 8 FPs and a precision of 0.886. 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. Overall, in the *active hang* setting out of 513 records of data, we had 443 cases of TN, 0 cases of FN, 62 cases of TP, and eight cases of FP. The accuracy of the approach, that is, correct assessments by the detector out all records, is 0.984. The main limitation concerns the false positives (discussed below).

Passive Hang. Similar patterns can be observed in Fig. 8c where predicted values mimic actual values. Surprisingly, there is no gap between predicted and actual value when injection is started around record 342. In this setting, we obtained 519 records of data, where we had 448 cases of TN, 60 cases of FN, 0 cases of TP, and 11 cases of FP, as show in Table V. Again, false positives are caused by a number of records where the magnitude of CPU spikes was predicted imprecisely.

TABLE V: 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

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. Accordingly, we state that the detection technique assessed in this technical report *is ineffective for those errors which simultaneously suppress the normative system activity and metric changes*.

It must be noted that an asymptomatic anomaly *does not* imply the lack of relevant lines in the log. To test this hypothesis, we scrutinized the logs of normal execution and passive hang. Our analysis reveals that the system generated error messages during passive hang – which do not occur in the normal run – reported in the following:

```
2015-09-28 13:00:14,610 INFO  [STDOUT] (WorkerThread
        #0[192.168.0.52:60983]) Timeout reached. java.net.
        SocketTimeoutException: Receive timed out

2015-09-28 13:00:14,613 INFO  [STDOUT] (WorkerThread
        #0[192.168.0.52:60983]) Socket closed java.net.
        SocketException: Socket closed
```

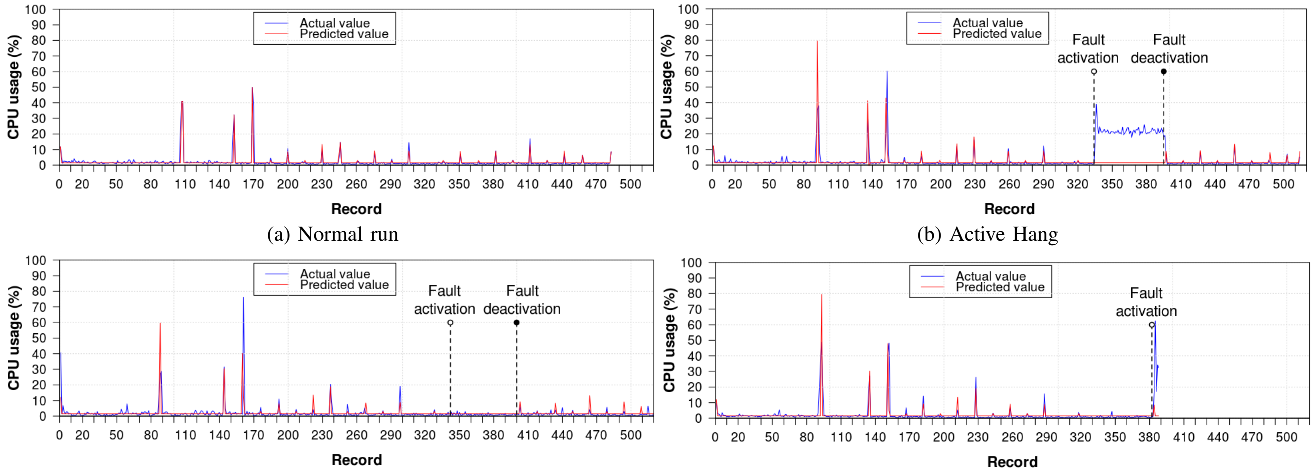
This observation suggests that asymptomatic anomalies can be reasonably addressed by *complementing* metrics-logs contextual detection with existing log-based failure analysis techniques. We believe this to be feasible in practice, given the wide body of literature on pinpointing error messages from logs or inferring models for conformance checking. An in-depth investigation of this proposition would exceed the scope of this report, and is thus left for future work.

Crash. In this setting, the prediction fairly resembles the actual values before the injection; the operation execution is aborted a few seconds after the injection around record 383 – see Fig. 8d. In this process, out of 387 records of data we had 372 cases of TN, 0 cases of FN, four cases of TP, and 11 cases of FP. Although the approach detected all the positive records, precision appeared to be low. This indicates the existence of too many false alarms in comparison to true positive alarms.

VIII. Improvements

A. Change Detection

The evaluation of the initial experiments into anomaly detection highlighted that the predicted values mimic the same pattern of actual values when the records are failure-free. However, in some cases, it did not correctly predict the magnitude of the spikes, leading to many FPs. For example, in



(a) Normal run

(b) Active Hang

(c) Passive Hang

(d) Crash

Fig. 8: Actual CPU usage versus predicted CPU usage for four separate runs: Normal, Active Hang, Passive Hang, and Crash with highlighted fault activation periods where present

Algorithm 1 - Change Detection

```
  - Anomaly Detection:
if ($|ac - pr| < \epsilon$) then
    $anomaly \leftarrow false$
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
    $anomaly \leftarrow false$
else
    $anomaly \leftarrow true$
end if
```

Fig. 8c (*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, as the actual value indicates 76.16% CPU usage while the predicted one indicates 40.14% CPU usage; moreover, both values are far bigger than the mean value of CPU usage, that is, 2.05%. Such a big difference led to the anomaly detection approach used to detect the obtained CPU usage spike, i.e., the actual value, as an anomaly, despite it being caused by regular activities of the ATC system. Therefore, a false positive alarm has been generated for that record. A similar pattern can be observed at record 94 in Crash run, Fig. 8d, where the actual value is 48.81% and the prediction is 79.47% – both indicating a spike on CPU usage but with significant gap between them, leading also to a false positive. This is because the criterion for detecting anomalous records used by the considered approach was based on evaluating the gap between the predicted value and the actual one being within the range of acceptable error of estimate, that is, 6.059 in our analysis.

The problem of modeling workload bursts or spikes in system monitoring, resource allocation and anomaly detection has been a topic of interest in recent studies [32], [33], [34]. 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 obtained 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 algorithm underlying the proposed policy is described in Algorithm 1, here referred as the *Change Detection* algorithm. Let us denote the actual value as ac, the predicted value as pr, standard deviation as σ, and standard error of estimate as ϵ. The Change Detection algorithm first checks whether the difference between ac and pr is within the accepted error of estimate, i.e., ϵ. If the condition is not satisfied then instead of reporting the record as an anomalous instance, it checks if both ac and pr have similar changes with respect to the standard deviation σ from the mean m. Otherwise, it reports the record as an anomaly. It worth to note that in most statistical based anomaly detection techniques, standard deviations (σ) from the mean are used as a threshold to detect significant deviation from normal behavior; often the values larger than $\pm 3.0\sigma$ are considered outliers [35]. The use of tighter thresholds than in the literature becomes possible in our setting, since we gain additional precision from analyzing the log context.

We then applied this new Change Detection algorithm on the records obtained by running the reference system with the three failure settings as described above. Table VI shows the results of this new analysis. It can be noted that there was a reduction of the FPs for all three experiments with respect to the ones obtained without Change Detection (Table V). The obtained FP reduction ranges from 50% of the *Active Hang* to the 73% of the *Passive Hang*. This translated into an improvement in terms of both Precision and Accuracy. Precision improved from 0.886 to 0.939 for *Active Hang* and 0.267 to 0.400 for *Passive Hang*. Accuracy improved from 0.984 to 0.992 for *Active Hang*, 0.863 to 0.869 for *Passive Hang*, and 0.972 to 0.984 for *Crash*. We conclude that our Change Detection algorithm provides an enhancement to the original technique in dealing with the magnitude of spikes.

TABLE VI: 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

B. Adaption of Time Windows

The anomaly detection results obtained from both the original technique (Table V) and the one enhanced with the Change detection algorithm (Table VI) were based on a default time window of 1 second. In our study, we observed possible delays between an operation action and its effect(s) becoming observable which may not be reflected in first second of time window. For instance, in the Passive Hang, Fig. 8c, the prediction based on the activities form log predicted a spike on CPU usage at 162th record, whilst the actual CPU spike occurred in the next record (163). Also, some of the operations may have a lasting impact for more than the default 1 second time window on CPU usage, such as the case on the 89th record in Passive Hang, where 59.57% is predicted, while the actual values indicate 26.98% for the current second and 28.69% for the next second. This is because, in the reference system an activity related to a service creation may take more than one second to be completed.

Therefore, we investigated if a further improvement in terms of accuracy can be obtained by enlarging the time window, that is, the observation period. We expanded the time window to a total of 3 seconds (±1 second from current time window) and re-ran our anomaly detection analysis. In this process, the detection of anomalies is decided with the relation to the status of previous, current, and next time window.

Table VII 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 the FPs slightly 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 V and Table VI), and the precision improved. For example, FPs reduced from 8 to 5 and from 6 to 1 for *Active Hang no CD* and *Crash w/ CD*, respectively, while precision improved from 0.984 to 0.990 and from 0.984 to 0.997, respectively. It should be noted that an additional expansion of the time window did not lead to further improvements of the results.

The obtained results suggest the larger time-window as another enhancement for the original technique, since it provides a further improvement of the detection accuracy by enlarging the observation period. In addition, the results in Table VII highlighted again that the use of the Change Detection algorithm is beneficial in terms of FPs reduction. For example, no FPs have been observed for both *Active Hang* and *Passive Hang* when the Change Detection algorithm is enabled and the time window is expanded.

TABLE VII: Anomaly detection results (3*sec* 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

IX. Threats to Validity

We briefly discuss the validity of the study based on the most relevant of the aspects listed in [36].

Construct validity. This study builds on experiments aiming to elicit stressful operations that require improvements for contextual anomaly detection. Our system is run with representative workloads provided by the industry vendor. While they are from real world scenarios, they may not match a particular scenario encountered in a specific deployment. We also rely on foundational statistics (such as Pearson correlation coefficient and regression models) and capitalize on the state-of-the-art in log analysis.

Internal validity. We used a mixture of metrics, diverse failures, and improvement strategies to mitigate internal validity threats. We assessed a wide range of resource utilization metrics, including CPU, memory and disk. The system was run with different failure settings to collect data records. We inferred a statistically significant correlation between CPU utilization and log activities, which provides a reasonable level of confidence in the analysis.

External validity. We ran the techniques on one example of a critical system from the ATC domain. However, the steps of the analysis should be applicable to similar datasets consisting of metrics and logs. Metrics are collectable by many established monitoring tools and logs are ubiquitously emitted by most applications/systems. The overhead entails the time required to establish event types from the logs. We addressed this step through POD-Discovery; moreover, several alternative log parsing methods exist (*e.g.*, SLCT, IPLoM, LogSig).

Conclusion validity. Conclusions have been inferred by replicating the analysis with different failure settings and by assessing several practical improvements. Our findings, supported by measurements on real systems and data, are useful to get an overall understanding of contextual anomaly detection, and its practicability and limitations for real-world systems. We are confident that the details provided should reasonably support the replication and generalization of our study by future researchers and practitioners.

X. Conclusion

In this technical report we analyzed the effectiveness of an anomaly detection approach based on log-metrics correlation from the literature [8] for a mission-critical air traffic control system. We selected this approach based on a literature review; previously, this approach has only been applied to cloud operations. 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, and discussed to complement the approach with the analysis of log behavior. The final evaluation detects anomalies with high accuracy and low delay. In future work, we plan to evaluate the approach in the presence of different fault injection campaigns, and a deep investigation of suitable techniques to complement the approach with behavioral log analysis, to detect asymptomatic failures like passive hangs.

XI. Acknowledgments

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] M. Farshchi, I. Weber, R. Della Corte, A. Pecchia, M. Cinque, J.-G. Schneider, and J. Grundy, "Contextual anomaly detection for a critical industrial system based on logs and metrics," in *Proc. 14th European Dependable Computing Conference, (EDDC)*, Sep. 2018.

[2] 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.

[3] 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.

[4] 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.

[5] H. Kang, X. Zhu, and J. L. Wong, "DAPA: diagnosing application performance anomalies for virtualized infrastructures," in *Proc. USENIX Hot-ICE Workshop*, 2012, pp. 1–8.

[6] J. P. Magalhes and L. M. Silva, "Anomaly detection techniques for web-based applications: An experimental study," in *Proc. IEEE Intl. Symp. on Network Computing and Applications*, Aug. 2012, pp. 181–190.

[7] 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.

[8] 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.

[9] A. J. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proc. 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2007, pp. 76–86.

[10] 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.

[11] I. Weber, C. Li, L. Bass, X. Xu, and L. Zhu, "Discovering and Visualizing Operations Processes with POD-Discovery and POD-Viz," in *Proc. DSN*, Jun. 2015, pp. 537–544.

[12] 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.

[13] T. Wang, W. Zhang, C. Ye, J. Wei, H. Zhong, and T. Huang, "FD4C: automatic fault diagnosis framework for web applications in cloud computing," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 1, pp. 61–75, 2016.

[14] T. Kelly, "Transaction mix performance models: Methods and application to performance anomaly detection," in *Proc. 20th ACM Symposium on Operating Systems Principles*. ACM, 2005, pp. 1–3.

[15] A. Pecchia, S. Russo, and S. Sarkar, "Assessing invariant mining techniques for cloud-based utility computing systems," *IEEE Transactions on Services Computing*, pp. 1–1, 2017.

[16] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, "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.

[17] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, Oct. 2004.

[18] M. Agyemang, K. Barker, and R. Alhajj, "A comprehensive survey of numeric and symbolic outlier mining techniques," *Intelligent Data Analysis*, vol. 10, no. 6, pp. 521–538, 2006.

[19] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: A survey," *Computer Networks*, vol. 57, no. 9, pp. 2093–2115, 2013.

[20] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *Proc. IEEE/IFIP Intl. Conf. on Dependable Systems and Networks (DSN)*, June 2016, pp. 654–661.

[21] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, 1966, pp. 707–710.

[22] Q. Fu, J. G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Proc. IEEE Intl. Conf. on Data Mining*, Dec. 2009, pp. 149–158.

[23] R. P. J. C. Bose and W. M. P. van der Aalst, "Discovering signature patterns from event logs," in *Proc. IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, Apr. 2013, pp. 111–118.

[24] W. Xu, "System problem detection by mining console logs," Ph.D. dissertation, University of California, Berkely, 2010.

[25] J. Stearley, "Towards informatic analysis of syslogs," in *Proc. IEEE International Conference on Cluster Computing*, Sep. 2004, pp. 309–318.

[26] R. Vaarandi, M. Kont, and M. Pihelgas, "Event log analysis with the logcluster tool," in *Proc. IEEE Military Communications Conference, (MILCOM)*, Nov. 2016, pp. 982–987.

[27] J. W. Osborne, "Prediction in multiple regression," *Practical Assessment, Research and Evaluation (PARE)*, vol. 7, no. 2, pp. 1–9, 2000.

[28] J. P. Onyango and A. Plews, *A textbook of basic statistics*. East African Publishers, 1987.

[29] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer, 2001.

[30] 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.

[31] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel, "Performance measures for information extraction," in *Proceedings of DARPA broadcast news workshop*, 1999, pp. 249–252.

[32] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proc. 1st ACM Symposium on Cloud Computing*. ACM, 2010, pp. 241–252.

[33] M. Sladescu, A. Fekete, K. Lee, and A. Liu, "GEAP: a generic approach to predicting workload bursts for web hosted events," in *Proc. 15th International Conference Web Information Systems Engineering (WISE)*. Springer International Publishing, Oct. 2014, pp. 319–335.

[34] A. Mehta, J. Drango, J. Tordsson, and E. Elmroth, "Online spike detection in cloud workloads," in *Proc. IEEE International Conference on Cloud Engineering*, Mar. 2015, pp. 446–451.

[35] A. Patcha and J. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[36] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction*. Kluwer Academic, 2000.