An Analytics-Based Method for Performance Anomaly Classification in Cyber-Physical Systems

Hugo Meyer[*], Uraz Odyurt[*], Andy D. Pimentel[*], Evangelos Paradas[†], Ignacio Gonzalez Alonso[†]
[*]Informatics Institute (IvI), University of Amsterdam, Amsterdam, The Netherlands
{h.d.meyer,u.odyurt,a.d.pimentel}@uva.nl
[†]ASML Netherlands B.V., Veldhoven, The Netherlands
{evangelos.paradas,ignacio.alonso}@asml.com

ABSTRACT

The analysis and correct categorisation of software performance anomalies is a major challenge in current industrial Cyber-Physical Systems (CPS). The automated evaluation of runtime performance metrics provides an overview of the software behaviour of CPS and may allow discovering or even predicting the occurrence of software performance anomalies. We present an approach to automatically identify deviations from expected performance behaviour of software processes running in a distributed industrial CPS. Our approach consists of collecting process performance signatures, using regression modelling techniques. This involves determining per process signatures and signature violations, enabling system anomaly type detection and classification. We evaluate different classification algorithms and predict the type of system performance anomaly that a signature violation can provoke. We demonstrate in our experiment that our design is capable of detecting and classifying synthetically introduced performance anomalies to real execution tracing data from a real semiconductor photolithography machine. Initial results show that we can achieve up to 93% of accuracy when classifying performance anomalies.

CCS CONCEPTS

• **Computer systems organization → Embedded and cyber-physical systems**; *Reliability*; *Availability*;

KEYWORDS

Automated performance anomaly detection, Industrial cyber-physical systems, Classification algorithms, Regression modelling

ACM Reference Format:
Hugo Meyer[*], Uraz Odyurt[*], Andy D. Pimentel[*], Evangelos Paradas[†], Ignacio Gonzalez Alonso[†]. 2020. An Analytics-Based Method for Performance Anomaly Classification in Cyber-Physical Systems. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3341105.3373851

1 INTRODUCTION

Nowadays, the domain of Cyber-Physical Systems (CPS) is one of the largest information technology sectors and CPS are present in many industrial applications, such as health, industrial automation, avionics, and military systems, amongst others. These machines typically are very complex distributed CPS, integrating a mixture of heterogeneous subsystems, including several nodes, interconnected via different types of networks. They are containing multiple dependent computing nodes with hardware and software components that perform different tasks, e.g., data processing, control, monitoring, logging and reporting, thereby realising a wide range of functionality and features.

The mentioned high integration of CPS in current industrial solutions is creating an increasing variety of complex computational systems that are tailored to specific scenarios and in some cases, even systems produced by the same manufacturer could have unique characteristics. The uniqueness of a specific CPS not only comes from the underlying system architecture, but also from different environments where they operate, different types of workloads that they manage, and different configurations applying to them. Considering this increasing complexity, numerous challenges arise with regards to functional and extra-functional operation of CPS. Classical performance analysis and performance anomaly detection techniques are facing major complications to model global and emergent behaviour of these systems.

As mentioned in [17], the emerging complexities in terms of management and configuration of CPS, urge the development of approaches to create self-adaptive systems, where human intervention is minimised. Moreover, a software performance anomaly could be translated to increased number of costly downtimes for the types of systems involved in high-tech industry, e.g., a semiconductor photolithography machine. To address these challenges, unexpected functional or extra-functional behaviour should be detected and corrected. However, the aforementioned uniqueness of each CPS added to the complexity of these systems is making online performance evaluation and performance anomaly detection almost infeasible through purely analytical, e.g., queueing theory, or purely experimental methods.

Our intention is to tackle the challenge of online software performance anomaly detection by analysing performance behaviour of software processes, e.g., CPU utilisation, occupied memory size, frequencies of exchanged messages. Any unexpected performance behaviour collected from the software components of an industrial CPS is a strong indication of the whole system behaving outside its expected boundaries. Many of these CPS perform highly repetitive tasks. For instance, a semiconductor photolithography machine is designed to perform collections of tasks over and over for a large number of wafers. Also, industrial printers normally have to schedule and print the same or similar content thousands of times. Hence, it is a valid expectation to have similar performance behaviour during the execution of similar tasks, as long as the system is performing correctly. This knowledge serves as a precursor for developing reference executions and metric readings, to be compared with future ones.

To be able to detect deviations from the expected performance behaviour, there is a need for comparative analysis of monitored performance behaviour against reference ones. In this paper, we present a methodology for automatic detection and classification of software performance anomalies in distributed industrial CPS with repetitive tasks. Figure 1 shows the complete workflow of our approach to create self-adaptive CPS, highlighting the main focus of this paper.

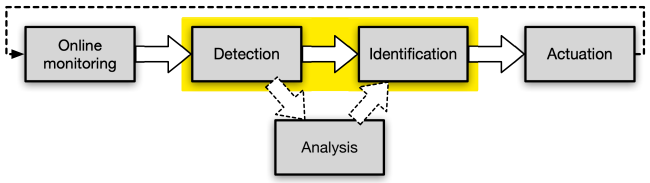


Figure 1: The high-level view of our performance anomaly identification workflow, including this paper's main focus, the identification step (highlighted in yellow)

Our approach can be considered as a form of analytics and thus involves two main stages, descriptive and predictive.

Descriptive analytics. With the purpose of creating a high-level but accurate representation of an industrial distributed CPS, we have developed a lightweight monitoring framework that is able to extract representative behaviour of a CPS by capturing information from the perspective of major communication libraries used by the software processes. The collected traces are used to drive a high-level simulation model of the CPS, based on discrete events and modelled behaviour is compared to real CPS behaviour [16].

Predictive analytics. The information collected in the previous stage is used to create baseline representations of active software processes. We create abstract performance representations of processes using regression modelling techniques, allowing us to collect insights regarding the process or processes responsible for anomalous behaviour, as well as their contribution. Process traces coming from different executions are collected and used to represent process behaviours belonging to normal and abnormal operation of the CPS. A database of system behaviours is created from the captured traces and classification techniques are trained to be able to detect abnormal behaviour while the system is operational.

Our approach sets the basis to perform *prescriptive analytics*, i.e., recommend corrective actions, while a CPS is operational. This is a step forward towards the creation of self-adaptive CPS. The presented methods and techniques are supported by experimental results based on traces obtained from real executions of ASML semiconductor photolithography machines.

The remainder of this paper is structured as follows: After providing a handful of important background information in Section 2, we will proceed to explain our methodology in Section 3. This will be complimented by our experimental results, elaborating achievements in Section 4. We will also provide information around the related literature in Section 5 and finalise with conclusions and future research lines in Section 6.

2 BACKGROUND

As a point of reference, here we will provide the basic notions and building blocks that we rely on throughout this paper.

2.1 Communication-centric monitoring

As part of the work for this paper, following the work done in [16], we have concentrated on a communication-centric perspective for our monitoring. Here, the goal of monitoring is to capture the behaviour of a CPS with enough detail. In communication-centric monitoring, this is achieved by collecting traces from purposefully planted probes in the code of the target system. Such an invasive probing provides us with enough data to support modelling and simulation efforts. This approach collects behavioural information for the whole system by only probing one or more major communication subsystems, passing messages between different software processes. Collected information include Extra-Functional Behaviour (EFB) metrics that are read and processed. In its final form, the tracing data consists of communication (read and write) and compute events. Communication events contain information such as, senders, receivers, message sizes, and timing information. Compute events contain information such as, CPU utilisation, CPU waiting times, process IDs and timing information.

A common choice for complex CPS is the deployment of communication subsystems, based on a publish-subscribe architecture. Semiconductor photolithography machines and radar systems can be considered as examples of such deployments. The aforementioned type of monitoring is well-suited for such communication subsystems and the traces obtained from this type of monitoring reflect the system behaviour with sufficient accuracy [16].

2.2 High-level modelling and simulation

We employ high-level modelling and trace-driven simulation of the CPS with the purpose of reasoning about the system behaviour. Accordingly, we have a detailed view of how different resources in the system are being used at different stages of the execution of a workload, e.g., CPU status and buffer utilisation. Moreover, the usage of high-level simulation models allows us to explore the effects of possible actuation mechanisms or countermeasures, to software performance anomalies in a safe environment before applying these actuations to the real system.

High-level modelling and simulation are rather intertwined notions, as a simulation is the execution of a calibrated model. The model is a simplified, but descriptive enough representation of the studied system, hence high-level. Having the knowledge of involved actor types, i.e., data producers (writers), data consumers (readers), and data brokers (communication libraries), complimented with interconnection information (topology), is sufficient to construct the high-level model. Calibrating this model for simulation is done by considering metric recordings, collected during the system's operation. All that information is available through communication-centric monitoring. The aforementioned tracing format, i.e., read, write, compute, is especially suitable for a discrete-event simulation. The high-level models and the trace-driven simulation will be covered in more detail in Section 3.3.2.

2.3 Software passports and signatures

There is a need for meaningful representations when it comes to system executions for various workloads. These representations should facilitate their storage and comparison of different executions. In general, representations of executions can be considered their *signatures*, while *software passports* are signatures of special cases, namely, reference executions. *Software passports* and *software signatures* focus on describing the EFB of software executions. Since the notions of software passports and software signatures apply to systems with repetitive nature, here we refer to an execution as a bounded segment of the overall operation of the system. These bounded segments, *phases*, can be in different granularities, namely atomic phases and combo-phases [18].

Signatures are generated by applying regression modelling techniques to EFB metrics captured over time.The comparison between a fresh execution and a reference one is based on goodness of fit tests. We consider such a comparison as a performance anomaly detection mechanism, which will also reveal the type of anomaly, based on the amount of deviation seen for goodness of fit tests. Anomaly types are derived from user composed conventions, as will be explained in Section 4.2.

2.4 Classification techniques

In this paper we consider that software performance anomalies can be of different types, for example *persistently harmless*, *persistently dangerous*, *transiently harmless*, and *transiently dangerous*. Automated and correct classification of executions is desirable with the purpose of providing online detection of software performance anomalies. Having enough known samples will allow us to benefit from different classification algorithms and identify the right category to which the execution at hand belongs. In this paper specifically, we have experimented with k-Nearest Neighbours (k-NN) [6], Linear Support Vector Clustering (Linear SVC) [1], Decision Tree (DT) [20], Random Forest (RF) [3] and Naive Bayes Classifier (NBC) [11] algorithms.

3 METHODOLOGY

According to Fowler and Rose, the main grand challenges in modelling and simulation of complex industrial CPS are: (1) the need to reduce the problem-solving cycles, i.e., time needed to design, collect information, build, execute, and analyse simulation models to support decision making; (2) development of real-time simulation-based problem-solving capability; and (3) interoperability between simulation models and CPS software, to have a complete cycle in place [10].

Although the mentioned challenges have been in the focus of the embedded and complex system design research community in the past, holistic approaches that cover monitoring, online representation, performance anomaly detection and activation in CPS are still missing. The development of self-adaptive systems requires the application of low-overhead system monitoring, high-level system modelling, trend prediction and online adaptation. Numerous research efforts still focus on bringing novel solutions in the aforementioned areas [7, 9]. However, few efforts have been made to combine all of these ingredients into a methodology that can set the basis for self-adaptive CPS. Moreover, CPS self-adaptation and optimal system resource utilisation has been in the focus of the scientific community as grand challenges. Our methodology focuses on addressing the grand challenges while keeping a holistic vision of cyber-physical systems.

3.1 An analytics-based approach to self-adaptive CPS

As depicted in Figure 2, our approach focuses on using descriptive, predictive and prescriptive analytics stages for a self-adaptive CPS.

Descriptive stage. Monitoring probes are used to capture the extra-functional behaviour of software processes running in the CPS. As mentioned before, a high-level, trace-driven, simulation model of the system is created, where software process behaviours are represented by traces of abstract events, and the hardware resources are modelled according to the CPS specifications. This simulation model allows us to perform system analysis in terms of resource utilisation and also to explore future behaviour of the system when testing suitable actuation mechanisms, such as changing process priorities or process scheduling policies.

Predictive stage. Simulation output traces corresponding to baseline executions are used to create software passports, intended to act as a signature construct for runtime performance behaviour of reference executions. During system operation, software signatures are dynamically created with the purpose of detecting abnormal performance behaviour of processes that can cause system-level performance anomalies [18]. As will be explained later on, regression modelling is used to create software signatures and passports for comparative purposes. The comparison reveals inconsistencies between the execution at hand (reflected by software signatures) and related software passports. Such inconsistencies are considered indicators for the presence of performance anomalies. In this approach, a comparison and classification engine, as shown in Figure 2, performs the online comparison of software signatures and passports, attempting to classify the type of detected performance anomalies. To this end, the classification engine has been trained by a database of training signatures that includes a large number of (labelled) performance anomaly samples of different types.

Prescriptive stage. Depending on the system-level performance anomaly predicted, different software-based actuation policies are explored and tested on a duplicated version of the aforementioned high-level simulation model. This high-level model is initially a copy of the model used during the descriptive stage. Although both models are synchronised, the one in the prescriptive stage is also modified to reflect applied actuation policies. Continuous feedback is given to the Actuation policy dispatcher, and once a suitable policy to address the predicted performance anomaly is found, it is applied to the CPS. Moreover, successful policies for different scenarios are saved for future use in the actuation policy database.

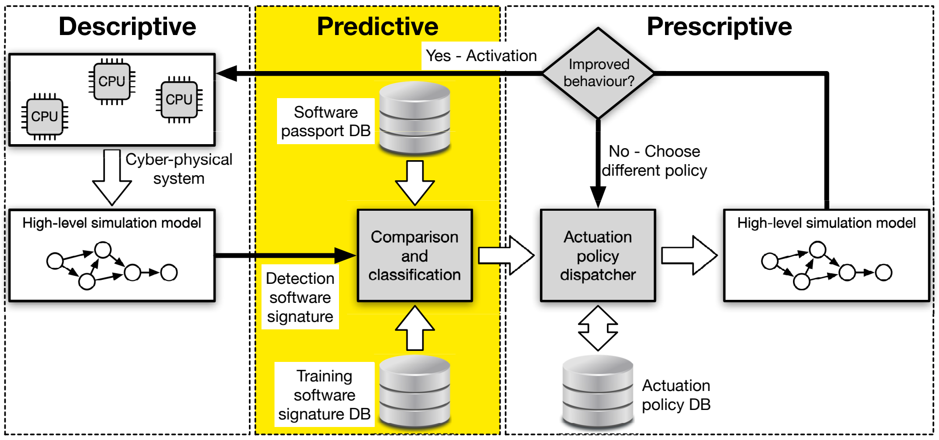

Descriptive
Predictive
Prescriptive
CPU
CPU
CPU
Cyber-physical system
High-level simulation model
Yes - Activation
Improved behaviour?
Software passport DB
Comparison and classification
Detection software signature
Training software signature DB
No - Choose different policy
Actuation policy dispatcher
High-level simulation model
Actuation policy DB

Figure 2: Self-adaptive CPS design, involving three main analytics stages, descriptive, predictive (this paper's main focus, highlighted in yellow), and prescriptive

Throughout the rest of this paper, it is important to distinguish between the two notions of software signatures used for training (training software signatures) and software signatures used for detection (detecting software signatures).

The aforementioned stages are continuous, which allows the creation of online reconfiguration policies for CPS, based on the history of behaviours and also considering new observations. In the following section, we will describe in detail the predictive analytics stage of our methodology, which is the main focus of this paper.

3.2 Predictive stage in detail

The passport and signature generation flows depicted in Figure 3 rely on traces collected from the CPS and in our case, we capture these traces from a communication-centric perspective, as discussed in Section 2.1. Traces are collected on a per process basis and then parsed, in order to create communication and computation events that are later normalised for consistency. For example, for send or write events, the patched traces contain a list of receivers, so the simulation engine can keep track of missing or delayed reads.

To create our software passports, reference executions are used and patched traces are simulated, after which the output is processed to extract performance related values such as, CPU utilisation, read and write counts over time, per process and per execution phase [18]. Different regression models are evaluated considering different goodness of fit tests, and the best regression function is selected to represent each performance metric for each process in a phase. These regression models allow us to track how the performance metrics of software processes evolve over time, which would not be possible if a simple average, minimum, or maximum value was considered to represent process behaviours.

To create training software signatures, a fault injection mechanism is used to synthetically modify the reference output, adding extra delays in specific events. This step is needed to collect data corresponding to different scenarios and corner cases that cannot be directly obtained from real machines due to the limitations of injecting faults in a running system. These anomalous traces are used as input to the simulator in order to assess the impact to performance metrics at the system level, e.g., execution time, read and write frequencies. Our fault injector implementation is capable of editing collected traces to introduce a chosen amount of delay to a chosen portion of processes at random. Modified traces need to be simulated since the cascade effect of event delays should be considered when analysing process behaviours. Then, once anomalous output traces are obtained, the same regression modelling techniques employed to create the software passports are used to create the training software signatures.

Software signatures are checked against software passports, evaluating the deviation of the goodness of fit coefficients. Evaluation is based on the percentage of deviation for the coefficient of determination (R^2) and for the Root-Mean-Square Deviation (*RMSD*) [15], as shown in Figure 3. We provide the formulas for R^2 and *RMSD* as a reminder, in Equations (1) and (2) respectively, such that

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \text{ and} \tag{1}$$

$$RMSD = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-3}}. \tag{2}$$

In these equations, $\hat{y}_i$ represents the estimated response for the ith observation, $\bar{y}$ represents the sample mean, y_i represents the observed response and n the number of data points. Note that $n - 3$ is the degree of freedom here, since these regression models have three parameters [15]. When it comes to detection, if we consider the comparison between software passports and signatures, y_i represents the data points collected for generation of the detecting software signatures, while $\hat{y}_i$ will be the values estimated by the software passports. Figure 3 depicts the details about passport and signature generation, as well as the classification flow.

Depending on the amount of detected deviation, and the type of performance anomaly injected, i.e., permanent or transient, the scenarios are labelled based on the projected effects. These effects are in the form of changes in process execution times. As it can be seen in Table 1, each row of the generated dataset includes deviations for each process, execution phase and metric combination, as

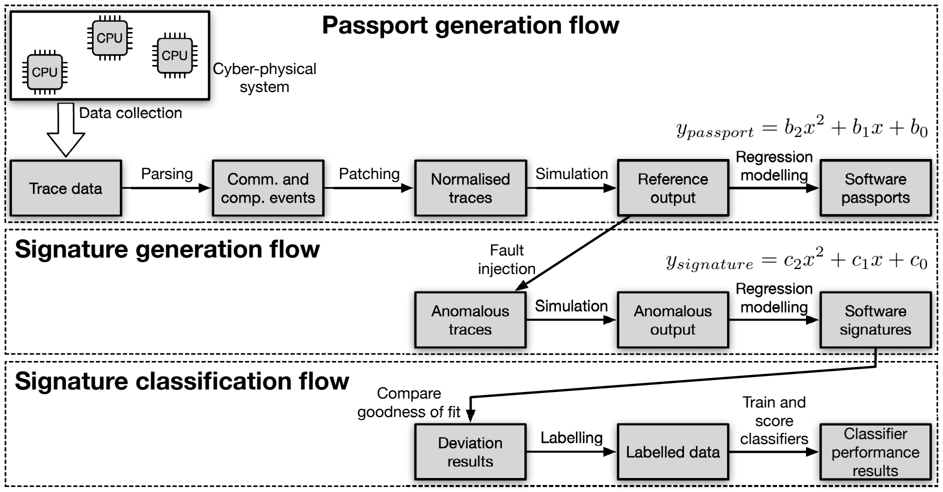


Figure 3: Steps involved in passport generation, signature generation, and signature classification flows

well as the assigned label. Different types of performance anomalies that we consider will be described later in Section 4.2. This dataset is used to create automated detection and anomaly classification techniques. The dataset is used to train different classifiers, which in turn are evaluated for their accuracy and timing, since this approach is aimed to work in an online set-up. These trained classifiers can be used for prediction of upcoming anomalous states, by classifying preceding phases, i.e., time windows.

The advantage of using classification techniques alongside regression and goodness of fit tests is that they allow for better response time. In this fashion, there will be no need for the phase to finish and the execution time to be known. Our approach is focused on obtaining combined performance behavioural patterns that could potentially lead to an anomaly. Additionally, we can have causality information (feature importance) by knowing the contributing PID-Phase-Metric combinations, which would have otherwise required the knowledge of the system's internals, e.g., critical path.

3.3 Implementation

The approach described in Section 3.1 is supported by four main software components, making up the backbone of three analytics stages and their respective tool collections. These components enable self-adaptivity and self-reconfiguration capabilities for CPS.

3.3.1 Lightweight communication-centric monitoring. To capture the trend of the system, our tracing tool follows a communication-centric approach, which allows us to limit the overhead introduced by invasive tracing [16]. The monitoring module is implemented at the system communication module, and when communication takes place, involved processes are indirectly traced. This approach introduces less overhead than fully invasive tracing of each and every separate software process at the cost of a reduction in the amount of information captured. The current information captured by our tracing tool includes per process data such as, event start and end timestamps, CPU utilisation, message sizes, message ids, messages sources and destinations, and memory utilisation.

3.3.2 Discrete-event simulation environment. The high-level simulation model of the CPS is trace-driven and consists of an application workload model and a hardware architecture model. The workload model is automatically generated based on the number of processes captured from the real system. This information is available through our lightweight communication-centric monitoring approach. The behaviour of processes in the workload model is automatically inferred from the collected traces and each simulated process executes, i.e., replays, one event at a time from these traces. Each time an event is executed, simulated versions of real hardware (CPU, memory, etc.) and software resources (communication libraries, etc.) are requested, with the purpose of simulating the competition for resources in the real machine. The simulation engine has been built using the OMNeT++ framework [21].

3.3.3 Fault injection mechanisms. Our fault injection implementation focuses on affecting the duration of events by modifying the end-time of events [18]. The software allows us to completely configure the percentage of processes to be affected, the percentage of events to be affected, the types of events (communication or computation), the type of performance anomaly to be introduced (transient or persistent), as well as the overlap between the processes selected to be modified, introducing transient or persistent anomalies.

3.3.4 Passport and signature generation. A complete ETL (Extract, Transform, Load) software pipeline has been created to extract per process, per metric and per execution phase information from the simulated traces. Regression modelling techniques are used to transform the extracted data into regression functions that are most accurately approximating the performance of processes over time. Another transformation takes place after comparing software passports against training software signatures, in order to generate a dataset where each row contains *RMSD* or R^2 deviation percentages for each process, phase and metric, as shown in Table 1. Data is then labelled according to the type of performance anomaly introduced and lastly, different classification algorithms, written in

Table 1: Different categories associated to scenario IDs, based on *RMSD* or R^2 deviation per metric, considering the categories, normal (0), persistently harmless (1), persistently dangerous (2), transiently harmless (3), and transiently dangerous (4)

Scenario ID	(PID-Phase-Metric)$_1$	(PID-Phase-Metric)$_2$	...	Label (anomaly category)
1	*RMSD* or R^2 deviation	*RMSD* or R^2 deviation	...	0-4
2	*RMSD* or R^2 deviation	*RMSD* or R^2 deviation	...	0-4
...				

Python, are trained. These classifiers will be used to detect if and when abnormal behaviour is being observed during runtime.

4 EXPERIMENTAL RESULTS

The results presented in this section aim to show the validity of our descriptive and predictive analytics stages, given in Section 3.1. First, we show examples of our software passports and signatures with the purpose of explaining how performance anomalies can be identified in advance by observing deviations in the goodness of fit test results for regression functions. We will follow this with performance evaluation of different classifiers during training and prediction, with the purpose of assessing our anomaly detection and classification engine.

The experiments performed during this study are based on a mixture of production and synthetic data. In order to create sufficient training scenarios, where different types of software performance anomalies could be assessed, we have used the fault injection mechanisms described in Section 3.3, introducing synthetic modifications to production traces.

4.1 Software passports and signatures

The creation of software passports, as explained in [18], considers a workload to be divided in different, but repetitive execution phases. In this set of experiments, we have considered one execution phase that corresponds to a repetitive task performed by an ASML semiconductor photolithography system.

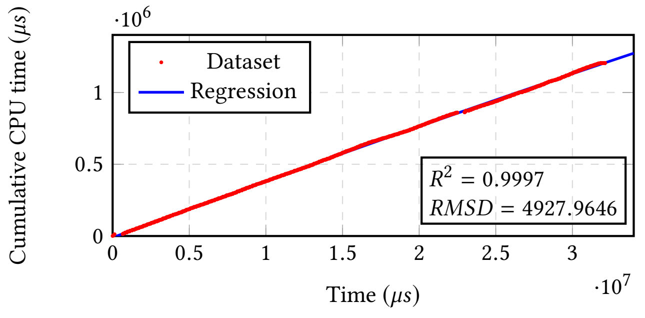
The simulation output traces, corresponding to the baseline execution, were used to generate software passports for each process, metric and experiment. Figure 4a depicts the cumulative CPU usage, of one process during one phase (dots in red), alongside the polynomial regression that approximates its trend. For this set of experiments, we have evaluated linear and polynomial regressions and selected the ones with the highest coefficients of determination (R^2) and the lowest *RMSD*.

In order to generate a proper dataset containing different types of performance anomalies, we have composed several fault injection scenarios, elaborated in the following steps,

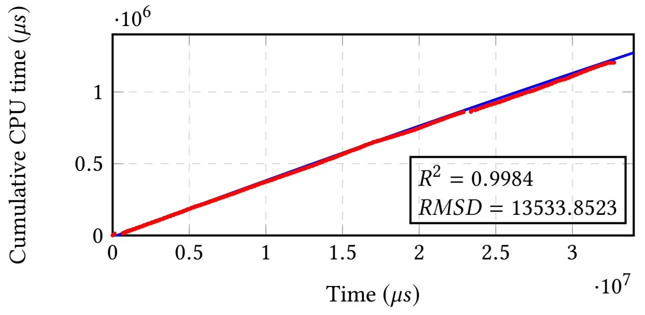
(1) From the total number of available processes, 15%, 30% and 45% of them were selected randomly to introduce modifications in their traces.
(2) From the processes selected in the previous step, two main performance anomaly groups, persistent and transient, were created. Software processes in the traces were randomly assigned to each group, while a 10% overlap between the two groups was ensured. For example, if we have the option to inject performance anomalies in twenty processes and we have to generate two groups with 10% overlap, the resulting groups will be,

$$\textit{Persistent group} = \{0, 1, 2, 5, 7, 8, 10, 11, 12, 13\} \text{ and}$$
$$\textit{Transient group} = \{0, 3, 4, 6, 9, 14, 15, 16, 17, 18\}.$$

The overlap percentage is configurable and its purpose is to create a more realistic scenario, where there is no clear separation between the set of processes that could cause persistent anomalies and the ones that could cause transient anomalies.
(3) Computation, read, write, or all types of events are selected from the designated group of processes in the previous step. For each of the event types selected, either 10%, 15%, 30%, or 45% of them are modified, adding some overhead to the event's elapsed time.
(4) The overhead applied to each event is either of 5%, 15%, 30%, or 45% of their total event elapsed time.



(a) Cumulative CPU time software passport



(b) Cumulative CPU time software signature

Figure 4: (a) Regression modelling result for a reference software passport, $(-6.447\text{e}{-11})x^2 + (3.996\text{e}{-2})x - 10523.271$, using cumulative CPU time, and (b) the corresponding violation

(5) Finally, the above steps were repeated five times to create sufficient data for different resulting categories.

Using the configurations described above, we created 1920 different scenarios, where each scenario contains traces corresponding to the monitored software processes. These modified traces were simulated, using the simulation environment described in Section 3.3, to determine the impact in the phase execution time. Figure 4b depicts the cumulative CPU usage software signature, obtained from one of the generated scenarios. The major difference with the passport shown in Figure 4a can be appreciated by observing the *RMSD* values.

4.2 Performance Anomaly classification

The signatures described before were created using a second-degree polynomial regression function and the *RMSD* deviation percentage was used as a feature for each PID-Phase-Metric combination (Table 1). The considered metrics in software passports and signatures, are cumulative CPU usage, cumulative writes and cumulative reads. With these metrics, we aim to correctly categorise the obtained signatures into five different categories, namely, normal, persistently harmless anomalies, persistently dangerous anomalies, transiently harmless anomalies, and transiently dangerous anomalies.

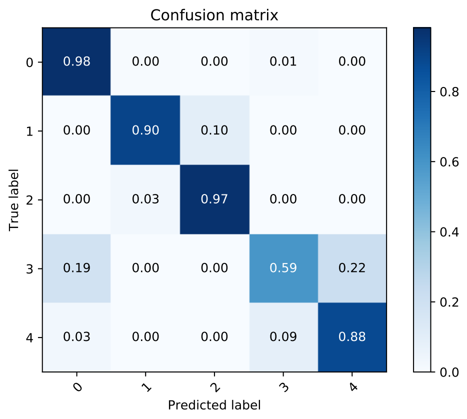
The original phase execution time is used as baseline and behaviour is considered to be normal when the observed execution time deviates between 0% to 2% from the reference execution time. When the observed execution time deviates between 2% to 4%, it is considered a harmless anomaly. Above 4% difference, the anomaly is considered dangerous for the operation of the CPS. Anomalies are considered persistent if the behaviour is repeated during several phases and they are considered transient if the misbehaviour occurs during a small set of phases.

The different metrics collected from different executions, together with the corresponding category, are used as input for five different classifiers, namely, k-Nearest Neighbours (k-NN), Linear Support Vector Clustering (Linear SVC), Decision Tree (DT), Random Forest (RF), and Gaussian Naive Bayes Classifier (Gaussian NBC)[1].

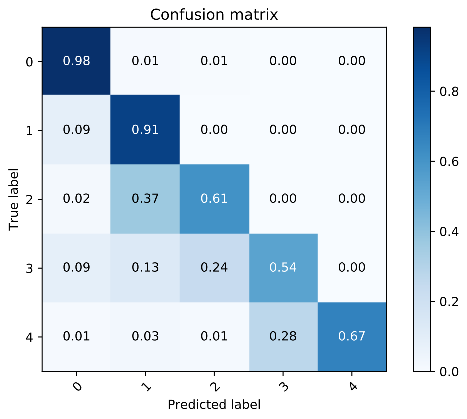
Three-fold cross validation was used to assess the score and timing behaviour of each classifier. Table 2 shows the obtained results for each classifier. As it can be observed, the two tree-based classifiers, DT and RF, outperform the other classifiers in terms of accuracy and scoring time. These two metrics are highly important to our set-up, considering the online performance anomaly categorisation needed to create self-adaptive CPS. Although RF seems to perform better in general, considering its score reaching to 93% of accuracy, the confusion matrices given in Figure 5 show that DTs, reaching to almost 91% of accuracy, make less errors when predicting labels. The confusion matrices are good indicators of which classes are misclassified and where hyperparameters can be tweaked to improve the accuracy according to the requirements of the use-case.

The set-up running the anomaly classification pipeline was executed on an Intel® Core™ i7-4770HQ processor at 2.20 GHz system

[1]The hyperparameters used with Scikit-learn library for each classifier are as follows: $n\_jobs = -1$ for k-NN; $C = 1.0$ and $max\_iter = 10000$ for Linear SVC; $max\_leaf\_nodes = 1000$ and $random\_state = 1$ for DT; $n\_estimators = 100$ and $random\_state = 1$ for RF; and default configuration for the Gaussian NBC.



(a) Decision tree classifier



(b) Random forest classifier

Figure 5: Normalised confusion matrices for (a) Decision tree classifier and (b) Random forest classifier, considering the categories, normal (0), persistently harmless (1), persistently dangerous (2), transiently harmless (3), and transiently dangerous (4) as True labels

Table 2: Accuracy and timing evaluation of classifiers

Classifier	Test score	Fit time (sec.)	Score time (sec.)
k-NN	0.899	0.082	0.361
Linear SVC	0.891	6.213	0.002
DT	0.906	0.014	0.001
RF	0.932	0.546	0.001
Gaussian NBC	0.762	0.005	0.004

with 8 GB of main memory, running Ubuntu 18.04.1 LTS. The deployed software infrastructure included Python 3.6.7, Scikit-learn 0.20.2 [19], and Luigi 2.8.3 [2].

It is also worth mentioning that the classification of a normal case has the highest accuracy. This is due to its frequency in the scenario pool, which is also the case in real-world situations. About 70%

of our scenarios are normal cases. We have also performed initial experiments to apply oversampling and undersampling techniques, resulting in lower accuracy for RF and DT. On the other hand, k-NN demonstrated high accuracy levels, even with oversampling and undersampling applied.

5 RELATED WORK

Anomaly detection and classification have been on the agenda of the research community for a long period. The relevant body of knowledge is fairly large, but is also covered rather well in two extensive survey papers. Both Chandalo [4] and Ibidunmoye [14] give collections of papers, diving into different aspects of anomaly detection. Numerous applications of anomaly detection techniques are given for intrusion detection systems, car fraud detection, mobile phone fraud detection, insider trading detection, medical applications, image processing applications, and text data applications, to name a few [4]. We are the first, as far as we know, to study online anomaly detection methods in the context of industrial CPS using concepts such as software passports and software signatures to identify deviations from baseline behaviours. The majority of the publications aiming at performance anomaly detection, given in [14], are indeed focused on distributed systems [13], cloud environments [12], and web application [5], domains.

When it comes to detection techniques, regression modelling is a recognised one [4]. We have based both our software passport generation and sample execution signature generation on regression modelling [18]. We are yet to utilise more complex techniques such as multivariate analysis, especially for regression modelling.

Classification-based techniques [8], involving training and testing steps have been used for anomaly detection before. Chandalo mentions a number of algorithms from the literature [4], but we have specifically considered five popular ones, k-NN [6], Linear SVC [1], DT [20], RF [3], and Gaussian NBC [11].

Our analytics-based approach involves grey box monitoring, as well as behavioural modelling and simulation. Most importantly, what separates our approach from the rest is its tailoring to industrial CPS, taking advantage of repetitive tasks over time and utilising the repetition metadata from the system.

6 CONCLUSIONS

In order to create self-adaptive CPS, the usage of descriptive, predictive and prescriptive analytics stages appears to be a promising path to follow. In this paper, we have shown that the creation of high-level models of CPS, the extraction of representative extra-functional behaviour, the comparison of baseline behaviour against runtime behaviour, and the usage of classification-based techniques to detect performance anomalies based on compressed representations of process performance readings, serve as a basis to develop autonomous CPS.

We have shown that the use of software performance passports and signatures can be advantageous for performance anomaly detection of systems with repetitive tasks. We have also described our experimental set-up and our anomaly detection method, based on process behaviour classification. Our initial experiments have shown that performance anomalies can be classified with 93% accuracy using per process and per metric goodness of fit deviations as features. We can conclude that software signatures in combination with classification techniques display a promising potential for comparative performance anomaly detection. Future work will focus on increasing the accuracy of current classifiers by enriching the collected signatures, considering other important metrics that determine process behaviour. We also plan on exploring the possibilities regarding the online re-training of classifiers, when new software behaviour is captured.

ACKNOWLEDGMENTS

This paper is composed as part of the research project 14208, titled *"Interactive DSL for Composable EFB Adaptation using Bi-simulation and Extrinsic Coordination (iDAPT)"*, funded by The Netherlands Organisation for Scientific Research (NWO).

REFERENCES

[1] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. 2002. Support Vector Clustering. *J. Mach. Learn. Res.* 2 (March 2002).
[2] E. Bernhardsson, E. Freider, et al. 2012. Luigi. https://github.com/spotify/luigi
[3] L. Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (oct 2001).
[4] V. Chandola, A. Banerjee, and V. Kumar. 2009. Anomaly Detection: A Survey. *ACM Comput. Surv.* 41, 3, Article 15 (July 2009).
[5] L. Cherkasova, K. Ozonat, , J. Symons, and E. Smirni. 2008. Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN).*
[6] T. Cover and P. Hart. 2006. Nearest Neighbor Pattern Classification. *IEEE Trans. Inf. Theor.* 13, 1 (Sept. 2006).
[7] P. Derler, E. A. Lee, and A. Sangiovanni Vincentelli. 2012. Modeling Cyber-Physical Systems. *Proc. IEEE* 100, 1 (Jan 2012).
[8] R. O. Duda, P. E. Hart, and D. G. Stork. 2000. *Pattern Classification (2Nd Edition).* Wiley-Interscience, New York, NY, USA.
[9] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra. 2007. A Framework for System-level Modeling and Simulation of Embedded Systems Architectures. *EURASIP J. Embedded Syst.* 2007, 1 (Jan. 2007).
[10] J. W. Fowler and O. Rose. 2004. Grand Challenges in Modeling and Simulation of Complex Manufacturing Systems. *SIMULATION* 80, 9 (2004).
[11] N. Friedman, D. Geiger, and M. Goldszmidt. 1997. Bayesian Network Classifiers. *Mach. Learn.* 29, 2-3 (Nov. 1997).
[12] S. Fu. 2011. Performance Metric Selection for Autonomic Anomaly Detection on Cloud Computing Systems. In *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011.*
[13] D. Gunter, B. L. Tierney, A. Brown, M. Swany, J. Bresnahan, and J. M. Schopf. 2007. Log summarization and anomaly detection for troubleshooting distributed systems. In *2007 8th IEEE/ACM International Conference on Grid Computing.*
[14] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth. 2015. Performance Anomaly Detection and Bottleneck Identification. *ACM Comput. Surv.* 48, 1, Article 4 (July 2015).
[15] R. Jain. 1990. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling.* John Wiley & Sons.
[16] H. Meyer, U. Odyurt, S. Polstra, E. Paradas, I. G. Alonso, and A. D. Pimentel. 2018. On the Effectiveness of Communication-Centric Modelling of Complex Embedded Systems. In *2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications.*
[17] L. Monostori, B. Kádár, T. Bauernhansl, S. Kondoh, S. Kumara, G. Reinhart, O. Sauer, G. Schuh, W. Sihn, and K. Ueda. 2016. Cyber-physical systems in manufacturing. *CIRP Annals* 65, 2 (2016).
[18] U. Odyurt, H. Meyer, A. D. Pimentel, E. Paradas, and I. G. Alonso. 2019. Software Passports for Automated Performance Anomaly Detection of Cyber-Physical Systems. In *Embedded Computer Systems: Architectures, Modeling, and Simulation.*
[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* 12 (Nov. 2011).
[20] L. Rokach and O. Maimon. 2005. *Decision Trees.* Springer US.
[21] A. Varga and R. Hornig. 2008. An Overview of the OMNeT++ Simulation Environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools '08).* Article 60.