Software Passports for Automated Performance Anomaly Detection of Cyber-Physical Systems *

Uraz Odyurt[1], Hugo Meyer[1], Andy D. Pimentel[1], Evangelos Paradas[2], and Ignacio Gonzalez Alonso[2]

[1] Informatics Institute (IvI), University of Amsterdam, Amsterdam, The Netherlands
{u.odyurt, h.d.meyer, a.d.pimentel}@uva.nl
[2] ASML Netherlands B.V., Veldhoven, The Netherlands
{evangelos.paradas, ignacio.alonso}@asml.com

Abstract. Software performance anomaly detection is a major challenge in complex industrial cyber-physical systems. The automated comparison of runtime execution metrics to reference ones provides a potential solution. We introduce the concept of software passports, intended to act as a signature construct for runtime performance behaviour of reference executions. Our software passport design is based on Extra-Functional Behaviour (EFB) metrics. Amongst such metrics, our focus has been especially on CPU time, read and write communication event counts of different processes. The notion of phases for systems with repetitive tasks during their execution and its fundamental role in our software passports has also been elaborated. We employ regression modelling of our collected data for comparative purposes. The comparison reveals inconsistencies between the execution at hand and the software passport, if present. Such inconsistencies are strong indicators for presence of performance anomalies. Our design is capable of detecting synthetically introduced performance anomalies to the real execution tracing data from a semiconductor photolithography machine.

Keywords: Software passports · Performance anomaly detection · Industrial cyber-physical systems · Regression modelling.

1 Introduction

There has been an increasing dependence on complex embedded systems within the industry. As every generation of microprocessors becomes more capable and more power-efficient than its predecessor, embedded nodes are deployed more than ever. Capabilities of embedded, i.e., purpose-built, computing power has allowed companies to develop increasingly complex systems involving heterogeneous elements, such as hardware architectures and operating systems, as well as many connected nodes. One major concern for any computing system is the

* 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).

occurrence of performance anomalies. These anomalies may be short-lived with negligible effects, may be constantly reoccurring and affecting the system, or may lead to an unresponsive state. Whichever the case, for anomalies to be avoided, there is a need for *anomaly detection*.

The analysis of performance behaviour and anomaly detection has become more and more time consuming, considering the sharp increase in the complexity of industrial cyber-physical systems. It is difficult to pin-point the actual anomaly, when it is happening, and where it is happening. This translates to increased number of costly downtimes for the types of systems involved in high-tech industry, e.g., a semiconductor photolithography machine.

It is our intention to tackle this challenge by developing methods and tools, facilitating automated monitoring and management of *Extra Functional Behaviour (EFB)*. In this context, EFB are the collection of behavioural metrics of a system, which are not part of its functional description, i.e., EFB are not encoded in software. However, there is an intimate dependency relationship between functional and non-functional behaviour, such that the non-functional behaviour is a consequence of the combination of functional behaviour, plus applied environmental variables. Input data is considered as an environmental variable as well. Examples of EFB metrics are CPU utilisation, occupied memory size, read and write frequencies, message sizes, delays, and lifetimes, amongst others. Throughout this paper, we will be considering CPU time, number of read events, number of write events, as well as durations of events and process executions as our main metrics to explain methods and demonstrate findings.

It is imperative to have mechanisms for anomalous performance behaviour detection in complex industrial cyber-physical systems, more than conventional computing systems. Any unexpected performance behaviour collected from the software portion of an industrial cyber-physical system is a strong indication of the whole system behaving outside its anticipated boundaries. In other words, the reliability of the whole system, depends on the reliability of its software. This is not an easy task. However, considering the operational specificities of these systems, there are angles to be exploited. Such types of systems are highly repetitive in the tasks they perform by definition. For instance, a semiconductor photolithography machine is designed to perform collections of tasks over and over for a large number of wafers. Another example is a radar system, applying same object detection workflows and algorithms over and over. It would be a generally valid expectation to have similar EFB metric values during the execution of similar tasks, as long as the system is performing correctly. This knowledge will serve us as a precursor to developing reference executions and EFB metric readings, to be compared with future ones. Such repetitive nature will allow us to divide the execution in compartments and consider clear and separate parts.

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. We introduce *software passports* as a representation of these references. Software passports can be considered in two types, static and dynamic. Further analysis will also provide clear insights regarding the process

or processes responsible for anomalous behaviour, as well as their contribution, since our software passport design includes separate regression models per every process involved in the execution. This root cause identification may be done in an automated fashion, if the comparison results are fairly clear-cut. We can summarise a high-level view of our approach towards anomaly detection using software passports in Fig. 1, including the optional analysis for identification of the root cause and a possible actuation step to rectify the issue. The whole workflow is to be continuously repeated. Our main focus in this paper has also been highlighted in the figure.

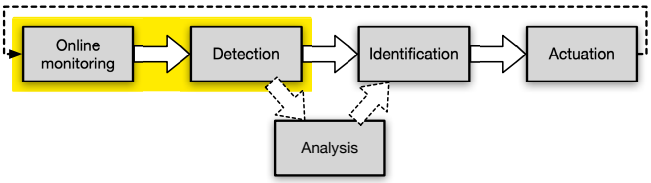


Fig. 1. The high-level view of our anomaly detection workflow, including the optional analysis for identification and this paper's main focus (highlighted in yellow)

Aside from the software passport's definition and its varieties, we provide the experimental results of our early prototype. We compare software passports generated based on reference runs, against executions involving deliberately injected anomalies. The following section provides background information and our industrial use-case. Section 3 describes our software passport design, continued by a description of our methodology in Sect. 4. After elaborating achieved experimental results in Sect. 5, we provide related work and our conclusions in Sect. 6 and Sect. 7, respectively.

2 Background and industrial use-case

The notion of performance anomaly is especially important for industrial embedded systems, as it is critical for these systems to deliver the functionality when it is being demanded. For instance, a semiconductor photolithography machine is designed to deliver a certain production yield and it has to fulfil such requirements, ideally at all times.

2.1 Performance anomalies

Before we continue, let us first define what performance anomalies and faults are. A *performance anomaly* is any *readily* detectable deviation in the system's performance behaviour. For instance, if a computational job takes significantly longer than it should to be fulfilled, a performance anomaly has occurred. Detecting the actual *fault* causing the performance anomaly however, requires *insight*

and analysis of the internal interactions of the system. As such, a performance anomaly is the result of a fault, but at the same time, not all faults will necessarily lead to a performance anomaly, as performance behaviour also depends on environmental variables. Accordingly, we define two types of anomalies, namely, transient and persistent anomalies.

Transient anomalies A transient anomaly is visible for a short period of time, or occurs only a few times. In such cases, either the fault is a short-lived one, or its impact to the overall execution is rather contained. For instance, with regards to timeliness, transient anomalies could be seen as delayed tasks in a contained part of the execution timeline.

Persistent anomalies In contrast to transient anomalies, a persistent anomaly is of the type that either will keep reoccurring, or will create cascading delays for all onwards tasks. This could be because of, for instance, dependencies between tasks. The overall cost to the execution is higher for such anomalies. Figure 2 depicts conceptual representations of transient and persistent anomalies alongside the comparison of their effect on the execution time. This is a rather simplified example and intends to convey the definitions of introduced constructs.

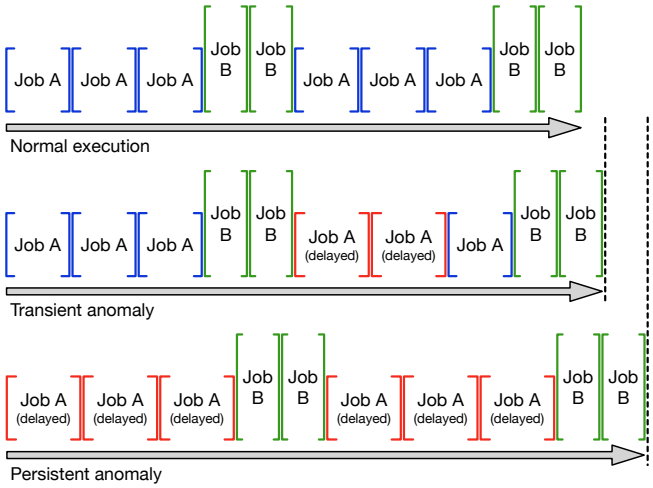


Fig. 2. The impact of transient and persistent performance anomalies in systems with repetitive tasks

2.2 Industrial use-case

Our industrial use-case and the primary source of our data is ASML's semiconductor photolithography systems. These systems are typical examples of com-

plex heterogeneous cyber-physical systems. They involve many computing nodes with different hardware architectures, running different operating systems and communicating via different messaging subsystems. Still, there is a close interdependency amongst different computing nodes, making these systems highly sensitive to untimeliness. In short, it is highly important to assure timely behaviour of different cyber elements in these cyber-physical systems, detect performance anomalies, and possibly react to them. We would like to reiterate that the most important criteria to be fulfilled here is the timeliness of different machine tasks, performed by multiple processes.

3 Software passports

Our software passport design is intended to act as a reference for executed processes to be compared against. Software passports include EFB metric recordings from previous executions. We consider two types of software passports, *static* and *dynamic*.

Static software passports Static software passports are generated from previous reference executions as post-mortem constructs. Any current execution that is supposed to match a reference one can be compared against the latter's software passport. The amount of deviation could potentially reveal performance anomalies. Interactions related to static software passports are depicted in Fig. 3.

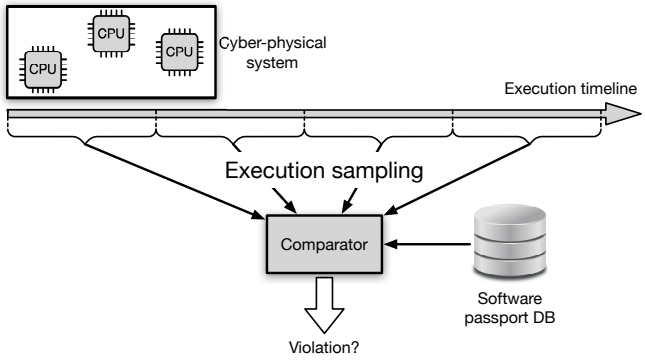


Fig. 3. High-level view of static software passports and their usage

Dynamic software passports Dynamic passports as shown in Fig. 4 and as the name suggests, are created in an online fashion, taking the immediate previous steps as references to the upcoming ones. That is, initial stages of an execution involving repetitive tasks could be used for passport generation and the following ones will be checked against them. This may also be in combination with previously generated static passports.

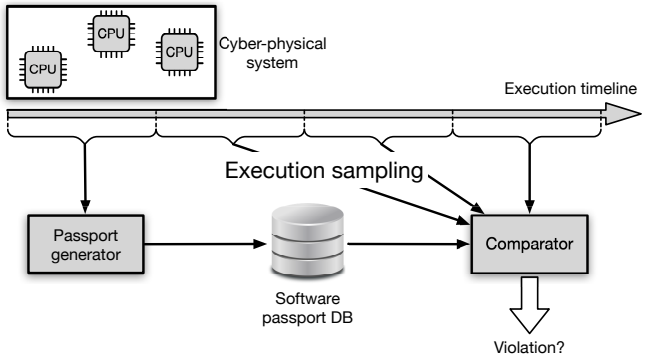


Fig. 4. High-level view of dynamic software passports and their usage

3.1 Execution phases

With that in mind, let us go through the concept of execution phases. A phase is considered as the duration in which the system is performing a recognisable and contained part of its overall job. Remember that we are dealing with a system that executes repetitive tasks, making the recognisability of phases even more tangible. Consequently, it is of great importance to capture initiation and termination of phases correctly. This is even more important for dynamic passports, as they are generated in an online fashion. Accordingly, a dynamic passport will detect repetition of phases and will consider the initiating ones as reference for the upcoming ones. The accurate knowledge of which initial phase, or phases are to be taken as reference depends on different factors.

At the same time, we consider phases as flexible constructs in the sense that one can decide how large, or more accurately, how long a phase should be. As an extreme example, one can consider a full execution from start to end as a single phase. Obviously, the choice of phase length has a direct impact on the generated software passport and its utility. Figure 5 shows a depiction of different *atomic phases* for different repetitive jobs, as well as larger *combo-phases* composed of a repeated pattern of atomic phases.

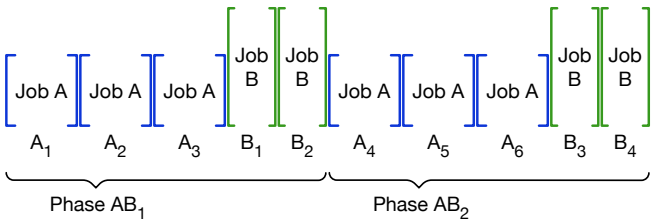


Fig. 5. Atomic and combo-phases

To put this into context, consider that a semiconductor photolithography machine's jobs are defined as batches of wafers. Each wafer could employ a different image for die exposure and these batches are all queued up in the machine's job queue. One can naively consider a whole queue of wafer batches, a single batch, a single wafer, or a single image exposure as phases from large to small, respectively.

A software passport can have different formats. Some of the considerations influencing this choice are types of collected data, i.e., EFB metrics in our case; phase structure, i.e., whether the passport is representing combo-phases or atomic ones; and gaps present in data point sets, i.e., how distant clusters of data points are from one another. We will mention two possible formats here, a passport including average, maximum and minimum values for a phase dataset, and a passport including one or more regression models. The immediate advantage of the latter is its compactness and relative sophistication. For our experiments, we have considered and built passports using the regression format. Depending on the distribution of points one can choose different regression modelling techniques. We have considered linear, multiple linear and polynomial regression models. Regression models can capture the trend of different metrics over time.

3.2 Data collection

An appropriate way of sampling arbitrary portions of an industrial cyber-physical system's performance behaviour is through EFB metrics. EFB metrics provide extra-functional performance readings of a system's execution, which are influenced by both functional behaviour and environmental conditions. Although EFB metrics provide a complete view over the system, the challenge present here is the sheer amount of available and collected data for this type of monitoring.

An efficient, but still complete, technique in order to track system behaviour is to collect monitoring data through major communication subsystems [11,10]. In this method, i.e., *communication-centric monitoring*, we are planting snippets as probes in the interface library code of the major communication subsystem. This library is used by many processes and thus, such an invasive probing allows us to capture calls from any user of the subsystem and provides us with details on communication and computation events. Though not a full picture of the system's operation, communication-centric monitoring provides sufficient detail on performance behaviour trends [11,10]. After applying necessary transformations on the collected traces, we will end up with traces composed of *read*, *write* and *compute* events, allowing us to perform a *replay simulation*.

4 Methodology

To understand our methodology for the experiments involved in this paper, we have to go through its main building blocks. These are namely, our simulation environment, our fault injection technique, and our prototype set-up.

4.1 Simulation environment

The aforementioned replay simulation closely resembles the actual system's behavioural trends and is interchangeable with it. Previously, we have shown [11,10] the power of communication-centric modelling of cyber-physical systems and how its deployment can facilitate the performance behaviour analysis of such systems. This facilitation is mainly about having valid performance trend detection, while reducing the monitoring effort by considering a smaller active portion of the system. As a result, a smaller, but impactful part of the system will be considered for EFB metrics collection, followed by performance trend detection.

Accordingly, collected traces will be used to generate a model of the system, to calibrate this model for conformance with the actual execution conditions, and eventually, to run the calibrated model as a replay simulation. The exact same monitoring performed on the actual system is also present within the replay simulation. This means that one of the outputs of our replay simulation is tracing data with the exact same format of the tracing data used for its calibration. That is how we validated the replay capability of our simulation.

There is also a need for presence of critical states in the system, to be able to demonstrate the potential of software passports in action. The replay simulation of the system will accept manipulated input to create scenarios resembling critical states. In other words, the replay simulation is to be used as a generator of monitoring data for fault injection. Thus, the output of the replay simulation can be considered as tracing data collected from a system with anomalous performance behaviour. The reason we are using the output of such a replay simulator and not the actual system for passport generation is that the fault injection cannot be done on the actual system at this point.

4.2 Fault injection

Fault injection can turn into a vast topic on its own rather quickly. Within the scope of this paper, we have focused on a specific set of fault injections. As we have explained before [11,10], events involve process idleness and CPU access delay. Duration of an event (t_E), which is the elapsed time from its initialisation (t_{ini_E}) till its end (t_{end_E}), is made up of CPU access delay ($delay_E$), CPU time ($comp_E$) and idleness ($idle_E$), such that

$$t_E = t_{end_E} - t_{ini_E} = delay_E + comp_E + idle_E.$$

We are considering synthetic increases to the duration of the event ($t_{E_{synth}}$), which means that we are increasing the combined duration of CPU access delay ($delay_E$) and idleness ($idle_E$), such that

$$t_{E_{synth}} = comp_E + (delay_E + idle_E + delay_{synth}).$$

Note that CPU access delay and idleness result from different conditions and we are not able to distinguish between them using the deployed tracing mechanism. While CPU access delay is simply a wait before CPU availability,

idleness could be a result of I/O waits, or functional and data dependencies to other processes. Nevertheless, our synthetic manipulation of traces does not depend on the distinction of the two, since we will be adding to the overall delay of an event, i.e., anything other than $comp_E$.

4.3 Prototype set-up

To assess the detection potential of our software passports, we have designed two experiment workflows, one for generation of software passports, and the other to demonstrate their usage as a reference. An overview of our workflows can be seen in Fig. 6.

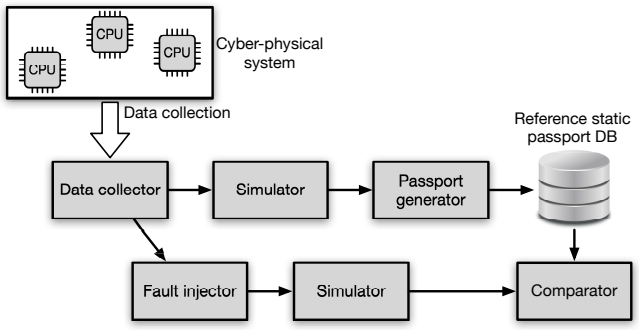


Fig. 6. Software passport generation and fault injection workflows

Passport generation workflow Considering the top flow given in Fig. 6, the data collector includes our communication-centric tracing techniques and tools [11,10]. What has been added here is the phase discovery. Each task of the system involves a number of processes. Considering the repetitive tasks we are dealing with, as explained in Sect. 3, we are able to detect per process start and end times for the set of events involved in processing of, in our case, different wafers. In this case, the detection is based on the dips in the CPU utilisation graph of the execution. For instance, a batch of ten wafers will have eleven distinctly observable dips, roughly representing transitions between wafers. This allows us to clearly separate each wafer's processing duration as a single phase.

After choosing one of the phases, the data representing this phase, which is part of the whole reference execution, is used as an input for our replay simulation. Same metrics are also collected from the simulation as output, which is extremely close to the real execution. The passport generator will consider per process cumulative CPU time, cumulative read events, and cumulative write events to generate regression models for the software passport. Usage of cumulative values in performance monitoring can be desirable, as it provides a monotonically increasing function.

Fault injection workflow As shown in the bottom flow of Fig. 6, fault injection is applied on tracing data before executing a simulation based on it. The output of the simulator and execution times of processes are collected and compared to reference executions. If the execution time is significantly longer, the collection of points for per process CPU time, read events and write events will be checked against relevant software passports. We can check the goodness of fit for a software passport's regression model, given the collection of metrics in time as our observation. There are multiple techniques to check goodness of fit, but within the scope of this paper, we are using *coefficient of determination (R^2)* and *Root-Mean-Square Deviation (RMSD)* tests.

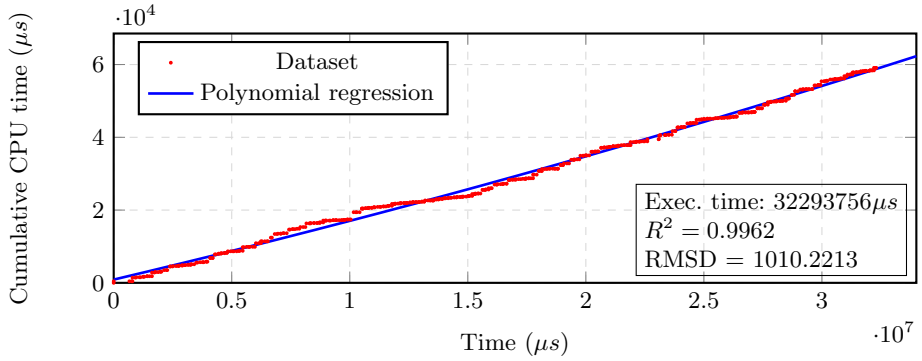
5 Experimental results

As indicated before, the experiments performed during this study are based on a mixture of production and synthetic data. The incorporation of synthetic data was a necessity as we needed a mechanism for controlled fault injection to test our developed techniques. Generation of such faults in a production system, on demand, is not certain and involves many limitations. 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. For instance, we can introduce 10% of delay to 20% of the processes involved in a phase. Depending on the delayed processes and their criticality, there may be different total phase delays introduced.

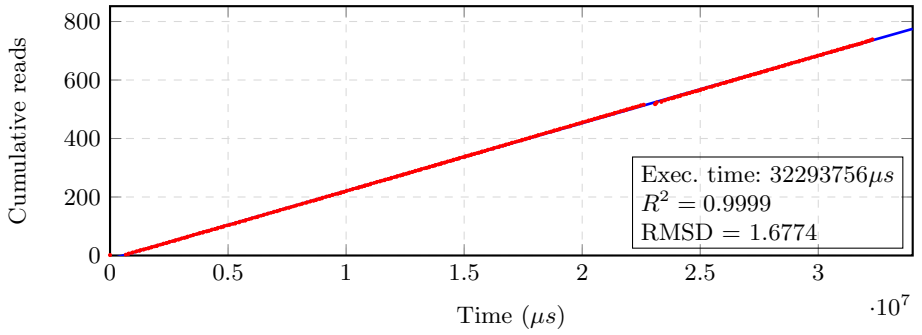
5.1 Detection results

In order to demonstrate our implementation of software passports and anomaly detection in action, we will present one of the processes from the tested phase. The detection includes two steps. First, the *length of the phase*, i.e., the execution duration of the phase, is checked. Note that the length of a phase should not be mistaken with the execution duration of a process in the phase, as the latter could be shorter. A 10% or above increase will be followed by a comparison against the software passport for that phase. The software passport includes three regression models per every process involved in the phase, for cumulative CPU time (depicted in Fig. 7a), for cumulative read events (depicted in Fig. 7b), and for cumulative write events.

These regressions' goodness of fit is checked against the current execution tracing data of the phase, after which a 5% or above difference between R^2 values, or RMSD values, will be interpreted as a violation. Figures 8a and 8b show the example process' CPU time and read event count violations compared to its passports, respectively. Note that there is no write event count violation, as this process is a consumer of data and does not perform any writes using the communication subsystem. Also, the library we use for regression model generation does support simple linear, multiple linear and curvilinear regressions.



(a) Cumulative CPU time



(b) Cumulative read event count

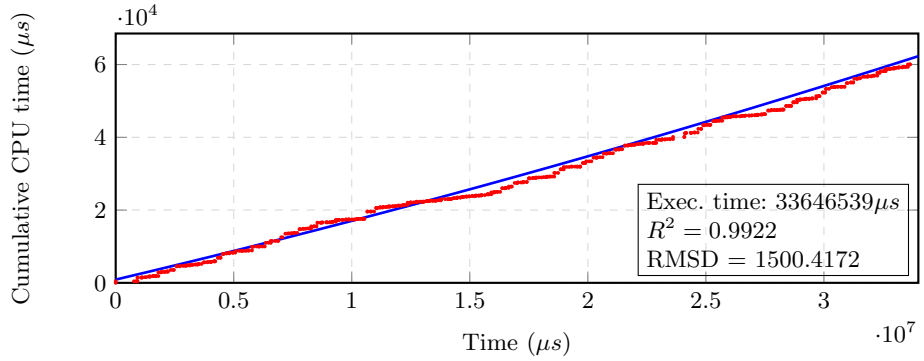
Fig. 7. Two software passports for an example process involved in the tested phase, with polynomial regressions, $(7.994\text{e}{-12})x^2 + (1.534\text{e}{-3})x + 890.944$ and $(-7.755\text{e}{-15})x^2 + (2.343\text{e}{-5})x - 12.813$, for cases (a) and (b), respectively

The choice will depend on the data, though linear regression models are the most common [6].

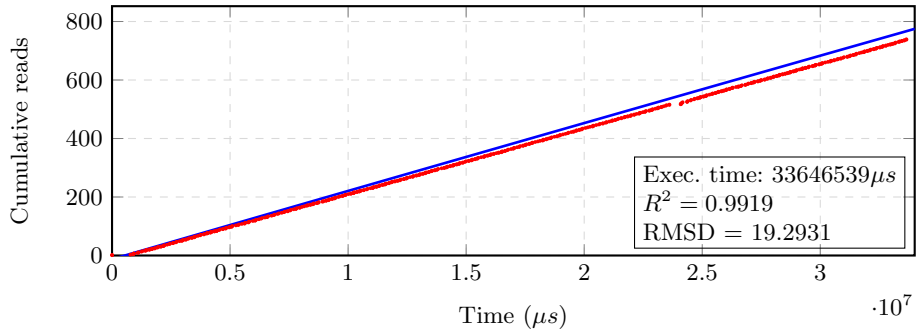
A comparison, based on regression models is especially useful for production environments, where the comparison happens in real-time. During online sampling, the comparison does not have to be postponed until reaching the end of the phase under scrutiny. Partial sampling, resulting in data points for a portion of the phase, will already be used for the goodness of fit check. The goodness of fit check provides us with the analysis capability, revealing the contribution of different processes, which is not detectable solely based on the execution time.

5.2 Towards identification

As it was shown in Fig. 1, to be able to perform identification, i.e., detect the root cause of the violation at hand, one might have to carry on with extra analysis. As we have separate passports per metric and per process in our design, it would be



(a) CPU time violation



(b) Read event count violation

Fig. 8. Software passport violations for different metrics of an example process involved in the tested phase

fairly straight forward to detect the violating processes. If the deviation in these contributing processes happens around the same time, or there is an obvious single initiator process with a large amount of deviation, we will have a clear-cut case. However, as there are usually functional and data dependencies amongst processes, interpreting violations will not be trivial. The link between metric violations and application-level behaviour of the system should be taken into account as well.

As part of our future work, there will be a need for a better understanding of behavioural variations and performing verifications, e.g., visual verification. A complex scenario could be simplified by breaking some of the irregular phases into smaller ones. Generating different non-linear regression models, such as isotonic regression, should also be considered.

6 Related work

The common use of regression models as a statistical tool for data estimation and inference is well argued in the literature [6,3]. This is especially true for different performance parameters of application processes as Lee and Brooks suggest [8]. Lee et al. also employ the notion of piecewise polynomial regression [9]. We have conducted a comparable strategy by dividing a timeline into meaningful phases, based on drastic changes in the values of CPU utilisation. Though, our division criteria aims at having meaningful and repeatable phases. We are not using all points from an execution, i.e., certain unsuitable parts are being omitted. Regression modelling has also been taken advantage of by Joseph et al. and Barnes et al., for correlating micro-architectural parameters with processor performance [7] and exploring parallel programme scalability [1], respectively. Regression modelling has been the choice of Torr and Murray [12], as well as Chen et al. [4], within the domain of image processing.

When it comes to anomaly detection in general, a comprehensive overview can be grasped by looking at two elaborate surveys by Chandola et al. [2] and Ibidunmoye et al. [5]. Our categorisation of anomalies as transient and permanent certainly fits the categorisation given in [5]. It is worth repeating that our focus for data collection is on EFB, which is arguably rather similar to the notion of Key Performance Indicators (KPI) used in other works. Another characteristic, separating our work, is its applicability to industrial cyber-physical systems, which by default involve repetitive tasks.

7 Conclusion and future work

In this work, we have shown that the use of software passports can be advantageous for performance anomaly detection of systems with repetitive tasks. We have argued in favour of a segmenting approach when characterising repetitive tasks and we have elaborated different phase constructs, as well as software passports based on them. We have also described our experiment set-up and detection results achieved using our prototype. We can conclude that though an initial version, software passports display a promising potential as a reference construct for comparative performance anomaly detection.

Our fault injection technique for anomaly introduction was based on manipulating simulations of the real system, also previously developed by us. This can be followed by considering fault injection within the actual system. A more complete software passport design is also part of our future work agenda. For instance, finding the right balance between the phase length, passport's detection capability, and computational effort for passport generation is the key for online deployment of dynamic passports. Additionally, not every change in performance behaviour of a system is an indication of anomalous behaviour. We would like to develop soft and hard limits for violation detection and interpretations based on them, leading to a more elaborate detection of anomalies. Another

lead we would like to follow is the categorisation of different types of anomalies, which will facilitate the decision making process when choosing effective actuation mechanisms.

References

1. Barnes, B.J., Rountree, B., Lowenthal, D.K., Reeves, J., de Supinski, B., Schulz, M.: A regression-based approach to scalability prediction. In: Proceedings of the 22Nd Annual International Conference on Supercomputing. pp. 368–377. ICS '08 (2008)
2. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. **41**(3), 15:1–15:58 (Jul 2009)
3. Chatfield, C.: Model uncertainty, data mining and statistical inference. Journal of the Royal Statistical Society. Series A (Statistics in Society) **158**(3), 419–466 (1995)
4. Chen, D., Shao, X., Hu, B., Su, Q.: Simultaneous wavelength selection and outlier detection in multivariate regression of near-infrared spectra. Analytical Sciences **21**(2), 161–166 (2005)
5. Ibidunmoye, O., Hernández-Rodriguez, F., Elmroth, E.: Performance anomaly detection and bottleneck identification. ACM Comput. Surv. **48**(1), 4:1–4:35 (Jul 2015)
6. Jain, R.: The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. John Wiley & Sons (1990)
7. Joseph, P.J., , Thazhuthaveetil, M.J.: Construction and use of linear regression models for processor performance analysis. In: The Twelfth International Symposium on High-Performance Computer Architecture, 2006. pp. 99–108 (Feb 2006)
8. Lee, B.C., Brooks, D.M.: Accurate and efficient regression modeling for microarchitectural performance and power prediction. In: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 185–194. ASPLOS XII (2006)
9. Lee, B.C., Brooks, D.M., de Supinski, B.R., Schulz, M., Singh, K., McKee, S.A.: Methods of inference and learning for performance modeling of parallel applications. In: Proceedings of the 12th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. pp. 249–258. PPoPP '07 (2007)
10. Meyer, H., Odyurt, U., Polstra, S., Paradas, E., Alonso, I.G., Pimentel, A.D.: On the effectiveness of communication-centric modelling of complex embedded systems. In: 2018 IEEE Intl Conf on Parallel Distributed Processing with Applications (ISPA). pp. 979–986 (Dec 2018)
11. Odyurt, U., Meyer, H., Polstra, S., Paradas, E., Alonso, I.G., Pimentel, A.D.: Work-in-progress: Communication-centric analysis of complex embedded computing systems. In: 2018 International Conference on Embedded Software (EMSOFT). pp. 1–3 (Sep 2018)
12. Torr, P.H.S., Murray, D.W.: Outlier detection and motion segmentation. In: Sensor Fusion VI. vol. 2059, pp. 432–443 (1993)