Measuring lead times for failure prediction

Frederico Cerveira*, Jomar Domingos†, Raul Barbosa‡, Henrique Madeira§
University of Coimbra, CISUC, Department of Informatics Engineering
**fmduarte@dei.uc.pt, jomar.domingos@student.dei.uc.pt†, rbarbosa@dei.uc.pt‡, henrique@dei.uc.pt§*

***Abstract*—Failure prediction anticipates system failures before they occur so that preemptive action can be taken, thus improving the dependability of the system. For effective failure prediction, the lead time, *i.e.*, the time between the occurrence of a fault and the appearance of a system failure, must accommodate both the prediction step and the preemptive action that is triggered after it. Lead time is intrinsically related to complex error propagation phenomena, which depends on the software architecture of the target system (*i.e.*, the system where failures are predicted) and on the dynamics of such software. For this reason, lead time is highly dependent on the specific nature and intrinsic details of the target system, which means that determining the distribution of lead time for a particular target system should be the very first step in developing failure prediction models. Furthermore, this step is of utmost importance, as it may decide whether failure prediction is viable for a given target system or not. For example, if lead time in a given target system is very short, it means that failure prediction is not viable in such system and classic (and expensive) fault tolerance should be applied. This paper proposes a method for obtaining the lead time distribution of a system using fault injection and presents a practical experiment illustrating such method for a virtualized system. The results suggest that the lead times of failures caused by software faults are usually much larger than those of failures caused by hardware faults.**

***Index Terms*—Failure prediction, Fault injection, Dependability**

I. Introduction

Failure prediction in computer systems, especially in complex systems such as high performance computing systems or systems that support the Cloud, has been a long quest of many decades. Unfortunately, practical solutions are still scarce and recent surveys/reflections on the topic still identify many gaps in the current state of the practice (*e.g.*, see [1], [2]).

The intensive use of artificial intelligence (AI) techniques for failure prediction has brought great expectations, especially in the last decade. But AI needs large quantities of failure data to provide effective results and real and meaningful failure data in computer systems is very hard to get. The use of fault injection techniques to produce realistic failure data for the development and evaluation of failure prediction models was first proposed in [3]. Since then, a large number of works have been published exploring this idea (*e.g.*, see [4], [5]).

However, one aspect that has been largely ignored in the fault injection studies used to collect failure data is the assessment of the failure behavior of the target system (*i.e.*, the system where failures are predicted) in the time domain. In particular, the assessment of failure lead time (*i.e.*, the time between the occurrence of a fault and the failure manifestation at system level) is essential to evaluate the feasibility of failure prediction in a given computer system. If the lead time is usually very short in that particular system, it could be impossible to predict failures and perform preventive actions in time to avoid system failures.

Lead time is highly dependent on the target system architecture and intrinsic details such as the software installed and its configurations, since lead time results from the series of complex error propagation events generated as consequence of a fault. Thus, it is of utmost importance to understand not only the general lead time distribution in the target system, but also to understand how lead time depends on the actual component of the target system affected by the fault or the type of faults.

In this paper, we propose the use of fault injection to assess the distribution of lead time of the target system as a mandatory first step in developing failure prediction solutions. Knowing the lead time distribution and, particularly, the way such distribution changes with the target system component affected by the fault, the type of faults, or even system configuration, is essential to evaluate the feasibility of failure prediction in the target system, to assess the percentage of failures that are impossible to predict (*e.g.*, failures with lead time of a few milliseconds are hard to predict in practice), and to identify critical system components that are related to failures showing very short lead time in a consistent way.

The paper is organized in the following way: Section II presents the proposed approach, Section III describes the methodology used in our experiments, Section IV presents and discusses the results, Section V discusses the limitations of the method and Section VI concludes the paper.

II. Proposed Approach

The idea consists of defining specific fault injection campaigns with the goal of evaluating the lead time distribution of the target system. Fault injection is a mature technology that has been used in many contexts [6], [7].

The proposed method is straightforward: it consists basically of three steps:

- *i)* Definition of specific fault injection campaigns.
- *ii)* Injection of faults and collection of failure timing data.
- *iii)* Analysis and conclusions for failure prediction strategy optimization.

Although the method is straightforward, the definition of the fault injection campaigns must meet the following criteria to assure accurate assessment of lead time:

- *Representativeness* - Include both representative hardware and software faults to assure realistic error propagation phenomenon and assess lead time accurately.

- *Reachability* - Inject faults in the relevant components of the target system to assure representative failure coverage.
- *Instrumentation* - Collect timestamps of key events with minimal timing disturbance in the target system.
- *Failure detection* - Detect failure occurrence using a method that is independent (*i.e.*, external) from the target system to assure reliable failure detection.

These criteria are achievable with current fault injection technology. The practical example of the instantiation of the proposed method presented in the rest of the paper illustrates how the approach can be used to evaluate the lead time distribution in a virtualized system.

III. Experiment Methodology

The study was carried out using an existing fault injection setup. Although this experimental setup has been originally designed for the more general goal of studying the impact of faults in virtualized systems, the injection campaigns already prepared for this setup are considered adequate to illustrate the proposed approach, as they meet the criteria mentioned above for accurate characterization of lead time distributions.

A. Physical setup

The setup includes two different physical systems, as depicted in Fig. 1. One of the systems manages the experiments and monitors and collects the results, while the other system provides the environment and computing resources where the failure prediction experiments take place. The controller system is equipped with a Intel Xeon E5620, 12 Gb of RAM and a 1 GbE network card. The target system includes two Intel Xeon Silver 4114, 32 Gb of RAM and a matching 1 GbE network card. The disk images used by the VMs were made accessible through Network File System (NFS).

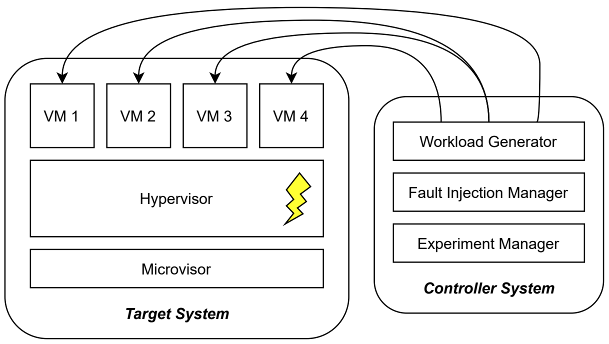

Fig. 1. Experimental setup used for the experiments.

A nested virtualization configuration was used with two layers of hypervisors (L0 and L1) and one layer of VMs (L2). Xen 4.11.1 was used for the L0 and L1 hypervisors, along with CentOS7 and Linux 4.14.89 for the L0 VM, Debian and Linux 4.19.0-6-amd64 for the L1 VM and CentOS 7 with Linux 5.5.11 for the L2 VMs. The L1 VM was configured with 2, 4 or 6 vCPUs and 12 Gb of RAM. The L2 VMs were configured with 1 vCPU and 900 Mb RAM.

B. Workload

The workload emulates a Solr server that provides access to a part (10 GiB) of Wikipedia's index. It includes only search operations and exercises the CPU, memory and I/O subsystems. The workload configuration was adapted to the resources assigned to the VMs and consisted of one client performing a new request each second.

C. Fault injection

Fault injection was used to emulate realistic transient hardware faults in CPU registers during the execution of hypervisor code, as well as software faults (bugs) in hypervisor code, such as those faults that are inserted during software development and escape testing. The focus on the hypervisor as target of the faults is justified, as the hypervisor is a critical element in typical virtualized systems. The ucXception fault injection framework[1] was used for both the injection of hardware and software faults. Fault injection in CPU registers followed the single bit-flip fault model [8] and targeted 15 registers, namely, RIP, RSP, RBP, RAX, RBX, RCX, RDX and R8 through R15. Software fault injection followed an extended fault model based on [9], which included the 12 operators listed in Table I.

TABLE I
Software fault model operators

Operators	Description
MFC	Missing function call
MIA	Missing `if` construct around statements
MIEB	Missing `if` construct + statements + `else` before statements
MIFS	Missing `if` construct and surrounded statements
MLAC	Missing `and` in logical expression used in branch condition
MVAE	Missing variable assignment with an expression
MVAV	Missing variable assignment with a value
WAEP	Wrong arithmetic expression in parameters of function call
WPFV	Wrong variable used in parameter of function call
WVAV	Wrong value assigned to a variable
WALR	Wrong algorithm – code was misplaced
WLEC	Wrong logical expression used as branch condition

Each fault operator was applied over the source code of four different files of Xen, including files responsible for managing memory and virtualization functions. The injected fault remains dormant until an explicit go-ahead is given, which is accomplished by encompassing the faulty code inside an *if* clause that depends on a Boolean variable. This variable is modified during runtime when the fault injection operation is ordered. Similar approaches have been used in past studies with the same purpose [6], [7].

D. Failure detection

Failures were detected by analyzing two different sources of data: i) a stream of pings to the various VMs; ii) the stream of request and responses of the Solr workload. A failure is detected whenever a ping request to any VM fails, when a Solr request is unanswered or when its response is incorrect.

This failure detection approach detects failures from the end user (*i.e.*f, the client) point of view, which is the relevant

[1]See https://github.com/ucx-code/ucXception

definition of failure for a failure prediction study (because the goal is to predict when the end user will experience a system failure). Possible (unlikely) internal component failures that do not propagate further to the workload are not considered, since they do not affect the service provided to the user.

E. Collected and derived metrics

During each experiment run, various metrics were collected for posterior analysis, which are used in this paper to infer the lead times of the studied system. These measurements were obtained with minimal instrumentation of the target system to limit intrusiveness and avoid impact on the accuracy of the obtained results, and include:

- *Moment when the fault injection was ordered* - For all the experiment runs, a timestamp value with the moment when the operation was issued is recorded.
- *Moment when the software fault is activated* - For SW fault injection, the moment when the software fault is first executed was recorded as a TSC value (*i.e.*, the number of CPU cycles since the system has booted).
- *Timestamps of steps in the experiment run* - Important goals during the experiment run had their timestamps stored. For example, the start and end of the run, moment when the VMs finished booting up, moments when the workload was started and stopped, *etc.*
- *Timestamp of first failed ping* - If a ping request fails, its timestamp is logged.
- *Timestamp of first failed workload request* - If a Solr request is not answered within the expected interval, or if its answer is incorrect, its timestamp is logged.

Lead time, which in this paper can be equated to the maximum hypothetical lead time obtainable by a perfect failure prediction classifier that relies on error symptoms, had to be calculated (see Figure 2) differently depending on the type of injected fault, due to constraints of instrumentation. For hardware faults, the lead time (LT_{HW}) is calculated by subtracting the moment when fault injection was ordered (t_1) to the moment of the first detected failure (t_4), as in Equation 1.

$$LT_{HW} = t_4 - t_1 \tag{1}$$

Whereas for software faults, the moment when the fault is activated (t_3) is subtracted from the moment when a failure is first detected, as shown in Equation 2. In this case, the moment of fault activation has to be converted from its native format, which is CPU cycles, to a timestamp. That conversion is obtained by combining information about the fixed TSC rate (*i.e.*, the amount of CPU cycles elapsed in 1 second), the average number of CPU cycles that it takes for the VM to boot up (obtained through a separate experiment) and the timestamp representing when the L1 VM was launched.

$$LT_{SW} = t_4 - t_3 \tag{2}$$

Due to the gap between t_1 and t_3, the lead times for hardware faults will be slightly overestimated. However, past experience tells us that, for hardware faults, this gap is very small and its impact on the results will be negligible.

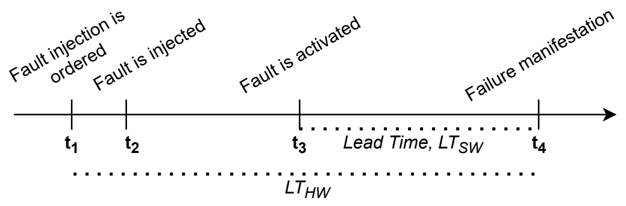

Fig. 2. Timeline of a fault injection run

IV. Experiment Results and Discussion

The analysis of the lead time distributions (*i.e.*, the main goal of the proposed method) is presented and discussed by answering the following two questions:

A Do failures caused by hardware and software faults show similar lead time distributions?

B For the failures caused by each type of fault, are there factors that affect lead time?

From the 1705 hardware faults injected, a total of 689 faults caused failures (*i.e.*, 40.4% of the HW faults caused failures), while the 1135 software faults led to 584 failures (*i.e.*, 51.4% of the SW faults caused failures)[2]. This relatively low effectiveness of fault injection is well-known and many fault injection experiments reported in the literature have shown similar percentages of faults that caused failures. It is worth noting that we injected realistic faults in order to guarantee realistic data propagation conditions and assure correct lead time assessment (non-realistic faults causing high impact on the target system to artificially increase the percentage of failures would lead to incorrect lead time assessment).

A. Do failures caused by hardware and software faults show similar lead time distributions?

Figure 3 compares the lead times of failures that were caused by hardware (HW) and software (SW) faults. It shows that **hardware faults tend to cause failures with lower lead times than those caused by software faults**.

The mean lead time of failures caused by hardware faults is 594 ms, whereas the corresponding value for failures caused by software faults is 54 065 ms. The lowest lead time seen for hardware faults was of 0.7 ms, while the lowest lead time for software fault injection was also a low 25 ms. The longest lead time when injecting hardware faults was 1200 ms, whereas software faults had a large outlier reaching 819 601 ms. Using CPU cycles instead of ms, failures caused by software faults had a mean lead of 1 740 686 million cycles, ranging between 1 473 253 million cycles and 2 529 090 million cycles.

Figures 4 and 5 provide a fine-grained view of the lead times for failures caused by hardware and software faults, respectively. From the point of view of the proposed method, these two figures represent the most important result to decide

[2]The dataset is available at http://eden.dei.uc.pt/~fmduarte/prdc21.zip

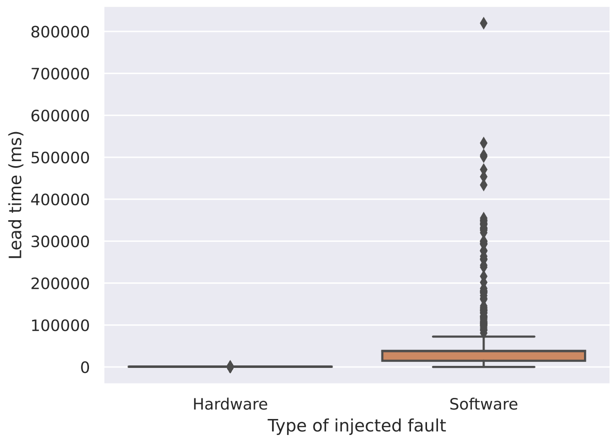

Fig. 3. Lead times for failures caused by HW and SW faults.

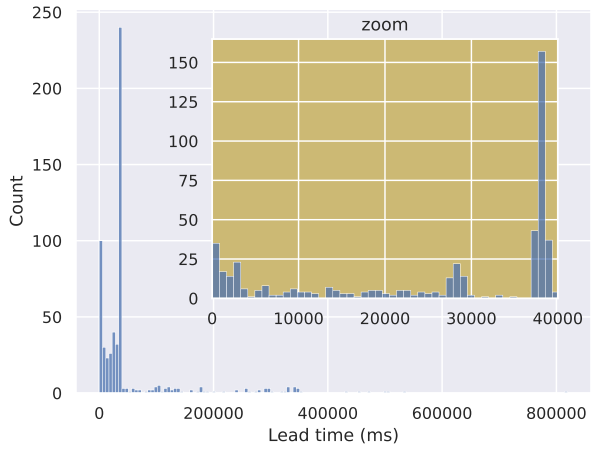

Fig. 5. Histogram of lead times for failures caused by SW faults.

about the feasibility of failure prediction in this particular target system. If the failure prediction model under consideration can process the input information (*i.e.*, symptoms) and output a prediction in the time frame defined by the lead time distribution, then failure prediction will be viable. Furthermore, these lead time distributions also allow estimating the percentage of failures that can be handled by a given failure prediction model, as the distribution can be used to show the percentage of failures with lead time higher than a given threshold defined according to the failure prediction model under consideration.

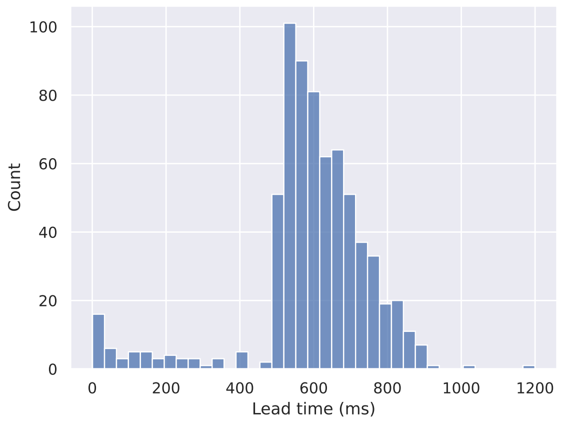

Fig. 4. Histogram of lead times for failures caused by HW faults.

Failures caused by hardware faults consistently show lead times in the range between 500 and 900 ms, with a small percentage of failures having even lower lead times. On the other hand, failures caused by software faults show a long tail distribution, with the majority of failures being concentrated in the 0 to 40 000 ms range (84% of all failures) and a considerable amount of failures having even higher lead times.

B. For the failures caused by each type of fault, are there factors that affect lead time?

Considering hardware and software faults separately, we can analyze different factors that may mold the failures occurrence and affect lead times, such as the component where the fault was injected (*e.g.*, which CPU register, which bit, which source code file or function) or the type of fault that was injected.

Figure 6 presents the lead times per CPU register. Since most registers in the x86 architecture have a specific purpose, it is possible that the lead times vary according to the affected register. However, according to the results it is not clear whether any specific register has an effect on lead times, as every CPU register shows similar lead times.

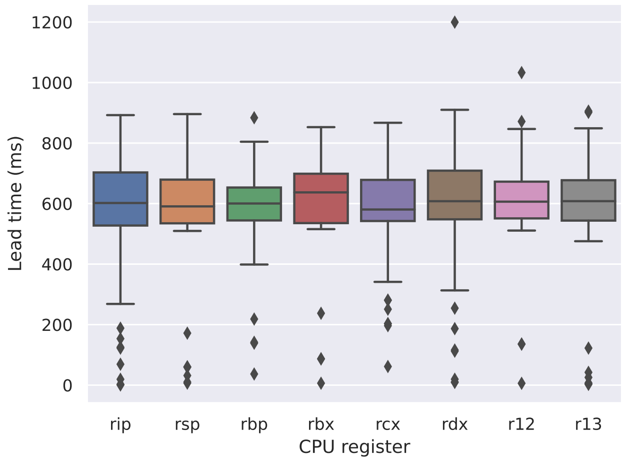

Fig. 6. Lead times per each CPU register.

A more detailed comparison uses both the CPU register that was affected by the fault and the bit that was flipped. Figure 7 shows a heatmap indicating the average lead time for each CPU/bit pair. Unfilled squares mean that no failure occurred.

Some patterns can be detected, namely in registers rip, rbp, rcx and rdx, where certain ranges of bits, such as the upper 32 bits, do not cause failures (light grey areas). However, in terms of lead time, there is no clear pattern.

The analysis whether different software fault operators (*i.e.*, different bug types) affected lead times is shown in Figure 8. The results show that some operators cause longer lead times.

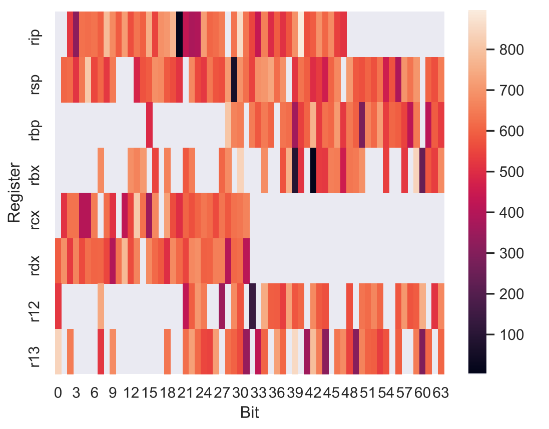

Fig. 7. Heatmap of average lead times for each CPU/bit pair.

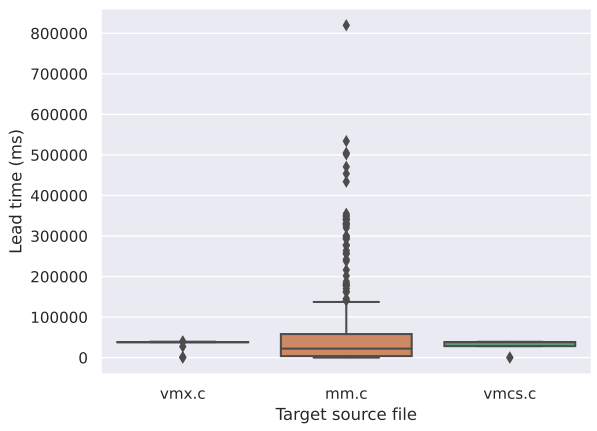

Fig. 9. Lead times per source file for failures caused by SW faults.

This is the case of MIEB, which has a much higher average lead time than the other operators. Moreover, the operators MIA, MFC and MIFS also exhibit a significant amount of outliers. This suggests that the type of software fault is highly relevant for failure prediction and some bug types are much more challenging, since the result is a much shorter lead time.

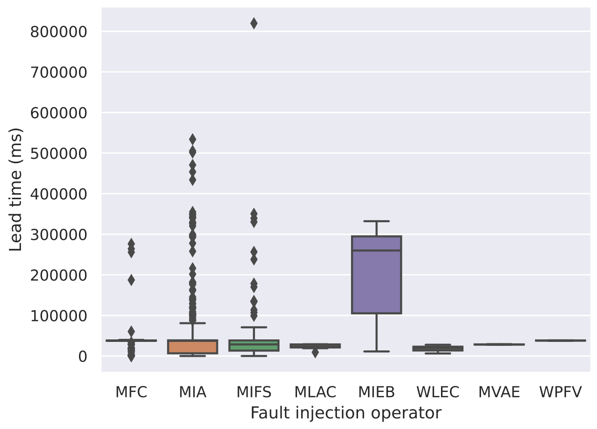

Fig. 8. Lead times per operator for failures caused by SW faults.

Figure 9 analyses the lead time per source file where the faults were injected and shows that the source file has a strong impact in the lead times. This is a very important result as it shows that some software components (in this case, different files of Xen) are much more critical than others. The file mm.c shows a different lead time distribution when compared to the others. Furthermore, the fourth file where the faults where injected is not shown in Figure 9 because none of the faults injected in such file led to failures. This result shows that the component affected by the fault has a dramatic effect on the failure probability and lead time.

V. Limitations

Although the presented experimental study shows the effectiveness of the proposed approach, the experiment itself contains several limitations that deserve discussion. These limitations result mainly from the fact that we used an existing fault injection setup, instead of defining and implementing a specific setup to illustrate the proposed method.

A first limitation is related to the lack of instrumentation to accurately trace the program execution and capture the exact moment when the injected fault was activated. Obviously, heavy instrumentation is highly intrusive, so the fact that we did not use such instrumentation has the advantage of not changing the timing behavior of the target system (due to the execution of extra code for instrumentation). The price to pay is to measure a lead time a bit longer than the real one. For the hardware faults we considered the moment when the operation was ordered, which is a bit earlier than the actual fault activation, while for software faults we record the exact CPU cycle when the fault was activated, but this value had to be converted to a timestamp, which reduces its precision.

A second limitation that reduced the precision of the results is related to the information available to detect failures, more specifically, their update rate. Two different sources were used (five ping streams to the VMs and Solr requests), but each of these sources was only updated about once a second.

A third limitation is related to the fact that faults were injected only in the hypervisor of the target system and the results just represent a partial view of the failure domain. Since the presented experiments have the main goal of illustrating the proposed method, we consider that this limitation is not particularly important. Obviously, the solution would be to perform more fault injection campaigns to cover other components of the target system, in addition to the hypervisor.

Despite the enumerated limitations, we believe that they do not invalidate the extracted observations for failure prediction in virtualized systems, especially because the focus on the key component (the hypervisor) means that the results refer to the weakest link in such systems, since failures in the hypervisor may have a very strong impact on all the VMs.

VI. Related Work

Some approaches have used Hidden Semi-Markov Models to predict failures [10], while recently machine learning approaches, such as using support vector machines or neural networks, are being researched [11].

The idea of using fault injection to generate failure data for failure prediction research was first presented in [3]. Irrera *et al.* proposed an approach using fault injection to accelerate data collection. To validate the data collected using this method, the authors developed an approach for estimating its representativeness [12]. João Campos *et al.* have also researched the usage of fault injection for failure prediction. In particular, they have developed guidelines for configuring the experimental setup as to reduce experiment run time while maintaining representativeness of the results [13]. They applied their research to a practical case study surrounding software faults in an operating system and were able to accurately predict failures using various machine learning techniques [5]. Although fault injection has been used in several works in the context of failure prediction, in all the previous works the focus was on the generation of failure data to train the prediction models or to evaluate the best symptoms. The relevant issue of studying the failure behavior of the target system in the time domain, with the goal of assessing the failure lead time distribution, was not, to the best of our knowledge, the focus of previous studies.

VII. Conclusion

We proposed the use of fault injection campaigns (following a specific campaign definition criteria) to assess failure lead time distributions as a first step of the development of failure prediction solutions. The paper has a double goal: i) present an experiment that shows the effectiveness of the proposed method and ii) present experimental results that provide an insightful view on the failure lead time distributions in a typical virtualized system, showing how such distributions are highly dependent on the type of fault and on the component of the target system affected by the fault.

Results show that the proposed method is effective and allow detailed characterization of the failure lead time distributions. This is essential to evaluate the feasibility of failure prediction in a given target system. Knowing that some types of faults cause failures with lead times of just a few milliseconds, while other types of faults show lead times of minutes is of utmost relevance to develop failure prediction models.

The results show that failure prediction in virtualized systems is better suited to predict failures caused by software faults than failures caused by hardware faults. The mean lead time of failures caused by hardware faults is 594ms, whereas the average lead time for failures caused by software faults is 54 065 ms. Furthermore, results show that software faults injected in some components (files) of the hypervisor show much shorter failure lead time than faults injected in other components. This means that the proposed method is able to identify the most critical target system components in terms of failure prediction lead time.

The low lead time of hardware faults may be explained by the importance that CPU registers, caches and memory have in terms of correct behavior of a computer. On the other hand, software faults, particularly those faults that escape software testing, tend to be activated only when complex conditions are met and may slowly corrupt the state of the application, as opposed to causing an almost instantaneous crash.

Acknowledgment

This work is supported by the FCT - Foundation for Science and Technology, I.P., through doctoral grant No. SFRH/BD/130601/2017 and within the scope of project CISUC - UID/CEC/00326/2020, by the European Social Fund, through the Regional Operational Program Centro 2020. This work is also supported by Project H2020 AI4EU (Grant Agreement 825619) and by the FCT within project ECSEL/0018/2019 and the ECSEL Joint Undertaking (JU) under grant agreement No 876852. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Czech Republic, Germany, Ireland, Italy, Portugal, Spain, Sweden, Turkey. Disclaimer: The views expressed in this document are the sole responsibility of the authors and do not necessarily reflect the views or position of the European Commission. The authors, the VALU3S Consortium, and the ECSEL JU are not responsible for the use which might be made of the information contained in here.

References

[1] D. Jauk, D. Yang, and M. Schulz, "Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice," in *SC '19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[2] J. Domingos, R. Barbosa, and H. Madeira, "Why is it so hard to predict computer systems failures?" in *Proceedings of the 17th European Dependable Computing Conference, 13-16 September*, 2021.

[3] M. Vieira, H. Madeira, I. Irrera, and M. Malek, "Fault injection for failure prediction methods validation," in *Proceedings of fifth Workshop on Hot Topics in System Dependability (HotDep 2009), DSN 2009*, 2009.

[4] I. Irrera, M. Vieira, and J. Duraes, "Adaptive failure prediction for computer systems: A framework and a case study," in *2015 IEEE 16th Int. Symposium on High Assurance Systems Engineering*, 2015.

[5] J. R. Campos and E. Costa, "Fault injection to generate failure data for failure prediction: A case study," in *2020 IEEE 31st Int. Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 115–126.

[6] D. Cotroneo, L. De Simone, P. Liguori, and R. Natella, "Profipy: Programmable software fault injection as-a-service," in *50th IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, 2020.

[7] E. van der Kouwe, C. Giuffriday, R. Ghituletez, and A. S. Tanenbaum, "A methodology to efficiently compare operating system stability," in *2015 IEEE 16th International Symposium on High Assurance Systems Engineering*, 2015, pp. 93–100.

[8] G. L. Ries, G. S. Choi, and R. K. Iyer, "Device-level transient fault modeling," in *Proceedings of IEEE 24th International Symposium on Fault-Tolerant Computing*, 6 1994, pp. 86–94.

[9] J. A. Duraes and H. S. Madeira, "Emulation of software faults: A field data study and a practical approach," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 849–867, 11 2006.

[10] F. Salfner and M. Malek, "Using hidden semi-markov models for effective online failure prediction," in *2007 26th IEEE International Symposium on Reliable Distributed Systems*, 2007, pp. 161–174.

[11] J. R. Campos, M. Vieira, and E. Costa, "Exploratory study of machine learning techniques for supporting failure prediction," in *2018 14th European Dependable Computing Conference (EDCC)*, 2018, pp. 9–16.

[12] I. Irrera and M. Vieira, "Towards assessing representativeness of fault injection-generated failure data for online failure prediction," in *2015 IEEE Int. Conf. on Dependable Systems and Networks Workshops*, 2015.

[13] J. R. Campos, E. Costa, and M. Vieira, "On configuring a testbed for dependability experiments: Guidelines and fault injection case study," in *Computer Safety, Reliability, and Security*. Cham: Springer International Publishing, 2020, pp. 419–433.