[Article] A survey on simulation-based fault injection tools for complex systems

(Article begins on next page)

A Survey on Simulation-Based Fault Injection Tools for Complex Systems

Maha Kooli, Giorgio Di Natale
Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier (LIRMM), France
maha.kooli@lirmm.fr, giorgio.dinatale@lirmm.fr

Abstract[1]—**Dependability is a key decision factor in today's global business environment. A powerful method that permits to evaluate the dependability of a system is the fault injection. The principle of this approach is to insert faults into the system and to monitor its responses in order to observe its behavior in the presence of faults. Several fault injection techniques and tools have been developed and experimentally tested. They could be mainly grouped into three categories: hardware fault injection, simulation-based fault injection, and emulation-based fault injection. This paper presents a survey on the simulation-based fault injection techniques, with a focus on complex micro-processor based systems.**

Index Terms—**Dependability, Faults, Fault Tolerance, Fault Injection**

I. Introduction

Reliability and availability are a main concern while designing electronic systems that are specially used in safety critical fields such as avionics, aerospace, military, and transportation. The uninterrupted performance and the progressive miniaturization of microelectronic devices in these type of systems make them more susceptible to be affected by external or internal faults. For these reasons, many researches focus on improving the dependability of the systems by inventing efficient techniques and methodologies.

Fault tolerance is one of the methods that permits to increase the dependability of a system. The goal of fault tolerant computing is to develop computing systems that perform correctly, respecting their functions, in the presence of faults. Several techniques and methods, software or hardware, are developed to tolerate faults on the systems.

Fault injection is a method for the assessment of fault tolerant systems. Different fault injection methods have been proposed in literature. They can be classified into:

- Hardware fault injection, where the actual hardware system is affected by external physical sources;
- Simulation-based fault injection, where the target system and the faults are modeled and simulated with a fault simulator;
- Emulation-based fault injection, where the target system is emulated (usually with FPGAs) and faults are injected in the emulator.

Nowadays systems are generally based on complex architectures with multiple micro-processors, memories, external devices and several layers of software (operating system, device drives, user applications) running on it. For these systems, fault injection is becoming a challenging task. This paper presents a survey on simulation-based fault injection techniques and tools that target these complex systems.

The rest of the paper is organized as follows. Section 2 presents the elements permitting to define the dependability of a system. Section 3 gives an overview of fault tolerance and some of its techniques. Section 4 introduces the fault injection method. Section 5 describes a set of efficient tools used to assess the dependability of micro-processor-based systems. Finally, section 6 draws some conclusions.

II. Dependability

Dependability is first introduced as a global concept that subsumes the usual attributes of reliability, availability, safety, integrity, and maintainability [1]. It represents the ability to avoid service failures that can happen to a system frequently and severely than acceptable. In other words, dependability is the ability to deliver products that can be trusted. It could also be defined as a measure of the system availability, reliability, and maintainability. Dependability can be determined and measured through three elements: attributes, threats, and means [2].

A. Attributes

The attributes represent a way to evaluate the dependability. They are represented by the following elements:

- *Availability:* It represents the probability that the system is available to operate correctly at a given instant of time. A highly available system is a system that will most likely be working at a given instant in time.
- *Reliability:* It represents the probability that a component in the system performs continuously in a predictable way without failure, i.e. for a prescribed time and under exact environmental conditions. A highly reliable system is a system that will most likely continue to work without interruption during a relatively long period of time.
  Reliability is defined over an interval of time rather than an instant in time, which is the case for availability.
- *Safety:* It represents the probability that the system either operates correctly or interrupts its functions in a way that nothing catastrophic happens to the users and the environment.

[1]This work has been supported by the joint FP7 Collaboration Project CLERECO (Grant No. 611404).

- *Maintainability:* It represents a measure of how easily the system can be fixed in case of failure. A highly maintainable system is a system that shows a high degree of availability when failures can be detected and repaired automatically.
- *Confidentiality:* It represents the absence of unauthorized disclosure of information.
- *Integrity:* represents the absence of improper system and data modifications.
- *Security:* It is a composite of confidentiality, integrity, and availability attributes [2].

Regarding these definitions, most of the attributes are more subjective and could not be measured. Only availability and reliability are quantifiable by direct measurements. For example, reliability can be measured as the failure over time, while safety is a subjective evaluation that needs judgmental information in order to have a level of confidence. That is the reason why techniques and methods developed to evaluate dependability are in reality to measure availability and reliability.

B. Threats

The threats represent the things that can touch to the dependability of the system. They are undesired and unexpected consequences resulting from non-dependability. They represent the following elements:

- *Fault:* It is a physical defect or imperfection that happens in the hardware, software or human component of the system. A fault can be the cause of specification mistakes, implementation mistakes, external disturbances, physical hardware component defects or misuses. A fault can be permanent, intermittent or transient depending on the length of time it persists.
- *Error:* It is a deviation of an external state of the system from the right service state. It could also be defined as a contradiction between the experimental or observed behavior, and the theoretically or expected behavior of a component in the system.
  Errors can be observed during a test session or using special mechanisms such as System Error Detection Mechanism (e.g., hardware exceptions handling, software checks, etc).
- *Failure:* It is the result of the deviation of the delivered service from the correct service. A failure is also defined as an instance in the time when the resulting behavior of the system does not correspond to the required specification.
  Considering the benefits provided by the system in absence of failures and the consequences of failures, failures can be classified into minor failures and catastrophic failures. Minor failures happen when the harmful results and the benefits provided by the right service delivery have similar cost. However catastrophic failures occur when the harmful results are extremely higher than the benefits provided by the right service delivery.

The relationship between faults, errors, and failures is represented by "the chain of threats" [2]. In fact, an error is the result of the activation of a fault, and a failure occurs when the error propagates to the service interface and becomes the reason of incorrect service. The failure of a component may be the cause of a permanent or transient fault in the system containing this component.

C. Means

The means are the techniques and methods that are able to increase the dependability of the system. They include the following techniques:

- *Fault Prevention:* It deals with avoiding the fault to occur on the system. It could be achieved by integrating development methodologies and good implementation techniques, such as design rules, modularization, use of strongly-typed programming languages, etc.
- *Fault Tolerance:* It means to prevent failures when faults are present in the system. The system remain working in an expected manner, according to its specifications in the presence of faults. More details are given in section III.
- *Fault Removal:* It means to decrease the severity and the number of faults in the system. It could be done either during the development phase or during the use of the system.
- *Fault Forecasting:* It is conducted by performing an evaluation of the system behavior with respect to fault occurrence or activation. Evaluation has two aspects: qualitative evaluation and quantitative evaluation.

The next section focuses on fault tolerance because it is the most powerful and famous technique permitting to increase the dependability of the system.

III. Fault Tolerance

Fault tolerance is a method to increase the dependability of the systems. It permits to avoid services failure in the presence of faults. Fault tolerant systems are defined as systems that are able to behave as expected and in a predictable way according to their specifications in the presence of faults. A fault tolerant system should be able to tolerate one or more types of faults.

Redundancy is a common strategy that can be used to tolerate faults that occur to a system. Redundancy is the use of additional hardware or functions that are not strictly necessary to functioning, but that are used in case of failure in other components. There are three types of redundancy:

- *Space Redundancy:* It contains the functional redundancy and the structural redundancy. *Functional Redundancy*, generally referred as a software method, is accomplished via additional functions. *Structural Redundancy*, generally referred as a hardware method, is accomplished via extra equipments added to the system in order to tolerate the loss or malfunctioning of some components.
  Space redundancy uses the technique of voting to increase the dependability [3]. For example, in Triple Modular Redundancy (TMR) (Fig. 1), three replica of the original module M are fed by the same input. If an error occurs

in one module among the three and no errors in the others, the correct result is obtained by the majority of the outputs. However, if two or more errors occur in two modules, three different responses are obtained and the voter cannot deliver a valid result. In this case, two solutions can be adopted: either the redundancy is increased in order to decrease the probability of such case, or the modules are re-executed, which is a more complex method that includes the space redundancy and the time redundancy.

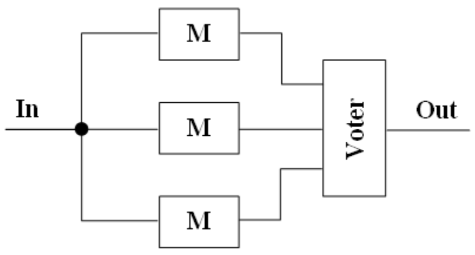

Fig. 1. Space Redundancy.

- *Information redundancy:* It corrects the cost of duplication or triplication of modules proposed by the space redundancy. As shown in Fig. 2, the information redundancy is based on coding the data so that the errors can be detected. Extra bits could be attached to the information to allow recovery from garbled bit.

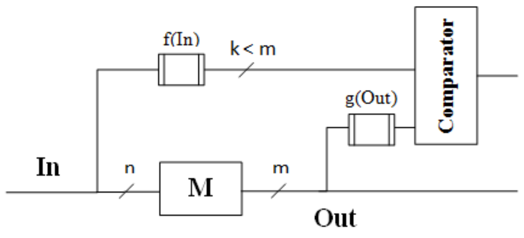

Fig. 2. Information Redundancy.

- *Time redundancy:* Conversely to the previous techniques, it does not replicate the original module. As shown in Fig. 3, it is based on re-executing the original function multiple times at different task moments and comparing the results of each one. It allows detecting transient or intermittent faults, but not permanent faults since they will always produce the same wrong output. The approaches of time redundancy are either with retry, where the failed instruction is repeated, or with rollback, where the execution is re-started from the beginning or from a checkpoint where a correct state has been saved.

Fault tolerance techniques can be implemented either in hardware or in software. The next 2 subsections present the main differences.

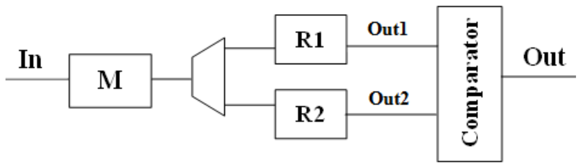

Fig. 3. Time Redundancy.

A. Hardware Fault Tolerance

Hardware fault tolerance techniques represent the techniques that change the hardware in order to tolerate faults. Although the hardware fault tolerant techniques are effective to improve the dependability of the system, they are usually costly because they affect the silicium area, the power consumption, and possibly the performance.

The most used techniques in hardware fault tolerance involve partitioning the computer system into modules that act as fault-containment regions. Each module is protected by redundancy so that, if it fails, others can assume its functions. In order to detect errors and implement recovery, special mechanisms and approaches are used, such as fault masking and dynamic recovery [1]. Fault masking is a structural redundancy technique that masks faults completely within a set of redundant modules. Dynamic recovery is used when only one computation copy is running at a time and it involves automated self-repair. It makes use of special mechanisms to detect faults. Dynamic recovery is generally more hardware-efficient than fault masking that uses voted systems.

B. Software Fault Tolerance

For micro-processor-based systems, Software Implemented Hardware Fault Tolerance (SIHFT) is the technique that permits to improve the dependability of the system at a lower cost without any modification in the target hardware [4]. SIHFT detects or tolerates faults in the hardware using software methods and without referring to any special hardware for error detection or fault tolerance. Several SIHFT techniques have been proposed in order to deal with different types of hardware errors such as Algorithm Based Fault Tolerance (ABFT) [5], Assertions, and Variable Duplication [6].

Software dependability issues can be faced on three different layers [6]:

1) Operating System Layer: In order to achieve a high level of dependability, the operating system can be modified. However, this method is very expensive because modifying the operating system requires high skills.

Regarding the operating systems in the most used devices like computers, there are no recent techniques that deal with the operating system layer because it is extremely complicated and there are other techniques that are easier to implement and that could lead to the same results. Only small and custom operating systems for embedded systems (used for instance in safety-critical applications) implement dedicated techniques to tolerate hardware faults.

2) Middleware Layer: The middleware layer permits to build an intermediate software layer that could intercept and modify all the communications between the operating system and the user application. This approach is very efficient when the user application is not modifiable and there is no possibility to access the source code of the application.

Interposition agents [4] are an example of non-application software programs that can transparently record and possibly alter the communication between the operating system code and the user code. The change of the environment, that is due to the modification of the system calls by the interposition agents, is not recognized neither by the software application nor by the operating system.

3) Application Software Layer: In this layer, the techniques proposed act directly on the software application to improve the dependability at this level. They represent a good solution when the application source code is available and can be modified. Some techniques that deal with the application software layer are presented in following.

Reliable code compiler (RECCO) for dependable applications [4][6][7][8] is a SIHFT technique proposed for checking memory errors. RECCO is based on a C/C++ source-to-source compiler able to increase the reliability level of a given application. This technique can be applied to any C/C++ source code. It is based on two strategies: re-order the code to decrease the probability of having fault on variables, and duplicate or triplicate the variables to detect or correct faults existing in the program data.

Control-Flow Checking via Regular Expressions [4][9] (CFCRE) is a SIHFT technique proposed for checking control flow errors. The check has inserted at source-code level using a signature methodology based on regular expressions. The signature checking is performed without dedicated watchdog processor but by resorting to Inter-Process Communication (IPC) facilities offered by most of the modern operating systems.

Algorithm Based Fault Tolerance (ABFT) [5] is a low-cost and highly efficient software-based resilience solution. ABFT is able to detect and correct transient and permanent errors in critical application data by exploiting, either redundant information inherent in numerical algorithms, or invariant relationships between data structures. The approach followed by ABFT is first, to encode the data using checksum schemes, then to redesign the algorithm to operate with the decoded data, and finally to distribute the computation steps in the algorithm among computation units.

IV. Fault Injection Overview

Fault injection is a powerful and useful technique to evaluate the dependability of the systems under faults [10]. It is defined as a validation technique of the dependability of fault tolerance systems based on the realization of controlled experiments in order to evaluate the behavior of the computing systems in the presence of faults. This technique can speed up the occurrence and the propagation of faults in the system in order to observe their effects on the performance of the system [11]. When setting up the fault injection environment, it is important to define the fault injection policy, such as fault location, injection time, fault duration, and the input data for the system.

A. Hardware Fault Injection

Hardware Fault Injection uses external physical sources to introduce faults into the system's hardware [12]. This means that the fault is injected in the real target system. There are two main groups of physical fault injection: injection with contact and without contact [11]. In the former, the faults are injected externally in pin level. However, in the latter faults are injected internally with heavy ion radiation or electromagnetic interference, which could be closer to a realistic fault model.

The advantage of hardware fault injection techniques is the ability to access some locations that are not easy to access by other techniques [11]. They are also suitable for systems requiring high time-resolution for hardware triggering and monitoring.

The disadvantages of such techniques is that the injection approach needs a special hardware and requires accessibility to the hardware of the target system, which may be sometimes not easy or very expensive to accomplish. Moreover, these techniques can present a high risk to damage the system under study. The results of the fault injection techniques in hardware are difficult to observe and to collect, which reduces the effectiveness of the method.

B. Simulation-based Fault Injection

In Simulation-based Fault Injection, the target system as well as the possible hardware faults are modeled and simulated by a software program, usually called *fault simulator*. The fault simulation is performed by modifying either the hardware model or the software state of the target system. This means that the system could behave as if there was a hardware fault [13]. There are two categories of fault injection: runtime fault injection and compile-time fault injection. In the former, faults are injected during the simulation or the execution of the model. In the latter, faults are injected at compile-time in the target hardware model or in the software executed by the target system.

The advantage of the simulation-based fault injection techniques is that there is no risk to damage the system in use. In addition, they are cheaper in terms of time and efforts than the hardware techniques. They also have a higher controllability and observability of the system behavior in presence of faults.

Nevertheless, simulation-based fault injection techniques may lack in the accuracy of the fault model and the system model. In addition, they have a poor time-resolution, which may cause fidelity problems.

Software Fault Injection is a special case of simulation-based fault injection where the target system is a large microprocessor-based machine that may include caches, memories, and devices, running a complex software. This technique is able to target applications and operating systems, which is not easy to do with the hardware fault injection.

C. Emulation-based Fault Injection

Emulation-based Fault Injection has been introduced as a better solution for reducing the execution time compared to simulation-based fault injection. It is often based on the use of Field Programmable Gate Arrays (FPGAs) for speeding-up fault simulation and exploits FPGAs for effective circuit emulation.

V. Fault Injection Environments

A wider and wider range of complex systems are nowadays used in safety-critical applications such as medical, space, automotive, and avionics. There is a tremendous need for the evaluation of the dependability of such complex systems. This section focuses on different tools that allow software fault injection in those systems.

A. FAUMachine

FAUMachine is a virtual machine that permits to install a full operating systems (Linux, Windows, DOS, Open- and NetBSD) and run them as if they are independent computers. This tool was developed in the Friedrich Alexander University of Erlangen-Nuremberg in Germany, and is available on [14]. FAUMachine is similar in many aspect to standard virtual machines like QEMU [15] or VirtualBox [16]. The property that distinguishes FAUMachine from the other virtual machines is its ability to support fault injection capabilities for experimentation. FAUMachine supports the following fault types [17]:

- *Memory Cells:* such as transient bit flips, permanent struck-at faults, and permanent coupling faults.
- *Disk, CD/DVD drive:* such as transient or permanent block faults, and transient or permanent whole disk faults.
- *Network:* such as transient, intermittent, and permanent send or receive faults.

FAUMachine did not permit injecting faults in the CPU registers yet. Bit flips could be easy to implement. Stuck-at faults is also possible but it is much more complex since FAUMachine uses just-in-time compiling.

In FAUMachine, the injection of fault could be done online via GUI, or defined (type, location, time, and duration of fault) via VHDL scripts.

Compared to existing fault injection tools, FAUMachine is able to inject faults and observe the whole operating system or application software. Using the virtualization, this tool provides a high simulation speed for both complex hardware and software systems [17]. FAUMachine also supports automated tests, including the specification of faults to be injected.

B. Jaca

Jaca is a fault injection tool that is able to inject fault in object-oriented systems and can be adapted to any Java application without the need of its source code, but only few information about the application like the classes, methods, and attributes names [10]. This tool was developed in the State University of Campinas(UNICAMP) in Brazil. It is an open source available on [18].

Jaca has a graphical interface that permits to the user to indicate the application's parameters under test in order to execute the fault injection [10].

Most of the fault injection tools are able to handle the injection of faults at low-level of the software. Jaca is different from the other tools in the fact that it can perform both low-level fault injection, affecting Assembly language element (CPU registers, buses, etc), and high-level fault injection affecting the attributes and methods of objects in a Java program [18].

C. LFI

LFI is a tool to make fault injection based testing more efficient and accessible to developers and testers [19]. LFI injects faults at the boundary between shared libraries and target programs, which permits to verify if the programs are handling the failures exposed by the libraries correctly or not. More in detail, LFI permits to automatically identify the errors exposed by shared libraries, find potentially buggy error recovery code in program binaries, and produce corresponding injection scenarios [20]. LFI is distributed through Sourceforge on [20].

Fault injection was rarely used in software development. LFI was developed in response to this. It permits to reduce the dependence on human labor and correct documentation, because it automatically profiles fault behaviors of libraries via static analysis of their binaries. The tool aims to provide testers an easy, fast, and comprehensive method to see how much the program is robust to face failures exposed between shared libraries and the tested programs [19].

D. Xception

Xception is a software implemented fault injection tool for dependability analysis. It uses the advanced debugging and performance monitoring features that exist in processors to inject realistic faults by software, and to monitor the activation of the faults in order to observe in detail their impacts on the behavior of the system [21]. Xception is a commercial software tool developed in the University of Coimbra in Portugal, and used in market since 1999.

Xception is a flexible and low-costly tool that could be used in a wide range of processors and machines (parallel and real time systems). In addition, it enables the definition of a general and precise processor fault model with a large range of fault triggers and oriented to the internal processor functional units [21].

E. RIFLE

RIFLE is a hardware fault injection tool where faults are injected in a pin-level of any module: the processor, the memory, the bus or other devices [22]. It is developed at the University of Coimbra in Portugal.

RIFLE is able to inject different types of faults and to analyze the impact of these faults on the system. Following some given criteria, RIFLE can define specific sets of faults through combining the fault trigger capabilities with versatile fault definition software. This feature represents an innovative feature of RIFLE.

The pin-level fault injection tools in the complex micro-processors generally make use of techniques such as prefetching, internal caches, pipelining, and delayed branches, which make the analysis of the results difficult. In addition, it is extremely difficult to observe the impact of pin-level faults injected in complex chips such as processors. RIFLE is the only pin-level fault injector that is able to perform analysis in order to observe the impact of faults on the processor [22].

F. LIFTING

LIFTING is a simulator able to perform both logic and fault simulation for stuck-at faults and single event upset (SEU) on digital circuits described in Verilog [23]. It is based on an event-driven logic simulation engine and performs a fault injection fault simulation. It was developed in the research laboratory LIRMM Montpellier in France. It has an objects oriented architecture and it is an open source tool available on [24].

LIFTING is different from other fault injection tools, because it provides many features for the analysis of the fault simulation results, which is meaningful for research purposes. In order to fault simulate complex micro-processor-based systems, it allows describing the hardware systems (including the memory), to define the software stored in the memory, and to inject faults in all elements of the hardware model.

VI. Conclusion

This paper gave an overview on the dependability concepts as well as the main techniques to tolerate faults in digital systems, either in software or hardware level. We presented a survey on fault injection techniques and environment, focusing on complex micro-processor-based systems running a software.

The increase of interest in electronic systems dependability was the motivation to develop modern fault injection methodologies and techniques. Apart from their basic feature which is increasing fault tolerance and improving the system dependability, these techniques have many other challenges, such as decreasing the technique cost, minimizing the time-to-market, and guaranteeing high efficiency of the results.

References

[1] K. Rozier, *Dependability management - Part 1: Dependability management systems, International Electrotechnical Commission*, IEC Std., 2003.

[2] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004. [Online]. Available: http://dx.doi.org/10.1109/TDSC.2004.2

[3] U. Raimund, R. Jaan, and T. V. Heinrich, *Design and Test Technology for Dependable Systems-on-Chip*. United State of America by Information Science Reference, 2011.

[4] G. Di Natale, “Software-implemented system dependability for safety critical applications,” Ph.D. dissertation, Politecnico di Torino, 2003.

[5] K.-H. Huang and J. A. Abraham, “Algorithm-based fault tolerance for matrix operations,” *IEEE Trans. Comput.*, vol. 33, no. 6, pp. 518–528, Jun. 1984. [Online]. Available: http://dx.doi.org/10.1109/TC.1984.1676475

[6] A. Benso, S. Di Carlo, G. Di Natale, L. Tagliaferri, and P. Prinetto, “Validation of a software dependability tool via fault injection experiments,” in *Proceedings of the Seventh International On-Line Testing Workshop*, ser. IOLTW ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 3–. [Online]. Available: http://dl.acm.org/citation.cfm?id=876896.880975

[7] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, and L. Tagliaferri, “Software dependability techniques validated via fault injection experiments,” in *Radiation and Its Effects on Components and Systems, 2001. 6th European Conference on*. IEEE Computer Society, Sept 2001, pp. 269–274.

[8] A. Benso, S. Chiusano, P. Prinetto, and L. Tagliaferri, “A c/c++ source-to-source compiler for dependable applications,” in *Proceedings of the 2000 International Conference on Dependable Systems and Networks (Formerly FTCS-30 and DCCA-8)*, ser. DSN ’00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 71–. [Online]. Available: http://dl.acm.org/citation.cfm?id=647881.737934

[9] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, and L. Tagliaferri, “Control-flow checking via regular expressions,” in *Proceedings of the 10th Asian Test Symposium*, ser. ATS ’01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 299–. [Online]. Available: http://dl.acm.org/citation.cfm?id=872025.872649

[10] R. de Oliveira Moraes and E. Martins, “Jaca - a software fault injection tool,” *Dependable Systems and Networks, 2003. Proceedings. 2003 International Conference on*, p. 667, June 2003.

[11] H. Ziade, R. Ayoubi, and R. Velazco, “A survey on fault injection techniques,” vol. 1, no. 2, pp. 171–186, July 2004.

[12] S. Ningfang *et al.*, “Fault injection methodology and tools,” *Electronics and Optoelectronics (ICEOE), 2011 International Conference on*, pp. V1–47 – V1–50, July 2011.

[13] G. A. Kanawati, N. A. Kanawati, and J. A. Abraham, “Ferrari: A flexible software-based fault and error injection system,” *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 248–260, Feb. 1995. [Online]. Available: http://dx.doi.org/10.1109/12.364536

[14] (2003-2013) Faumachine. [Online]. Available: www.FAUmachine.org/

[15] Qemu. [Online]. Available: http://wiki.qemu.org

[16] Virtualbox. [Online]. Available: www.virtualbox.org/

[17] S. Potyra, V. Sieh, and M. D. Cin, “Evaluating fault-tolerant system designs using faumachine,” in *Proceedings of the 2007 Workshop on Engineering Fault Tolerant Systems*, ser. EFTS ’07. New York, NY, USA: ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1316550.1316559

[18] G. M. L. Nelson. (2009, September) Jaca software fault injection tool. [Online]. Available: http://www.ic.unicamp.br/ eliane/JACA.html

[19] P. D. Marinescu and G. Candea, “Lfi: A practical and general library-level fault injector,” in *Proceedings of the Intl. Conference on Dependable Systems and Networks (DSN)*, Portugal, June 2009.

[20] Lfi: Library-level fault injector. [Online]. Available: http://sourceforge.net/apps/trac/lfi/

[21] J. Carreira, H. Madeira, and J. G. Silva, “Xception: A technique for the experimental evaluation of dependability in modern computers,” *IEEE Trans. Softw. Eng.*, vol. 24, no. 2, pp. 125–136, Feb. 1998. [Online]. Available: http://dx.doi.org/10.1109/32.666826

[22] H. Madeira, M. Z. Rela, F. Moreira, and J. G. Silva, “Rifle: A general purpose pin-level fault injector,” in *Proceedings of the First European Dependable Computing Conference on Dependable Computing*, ser. EDCC-1. London, UK, UK: Springer-Verlag, 1994, pp. 199–216. [Online]. Available: http://dl.acm.org/citation.cfm?id=645330.649779

[23] A. Bosio and G. Di Natale, “Lifting: A flexible open-source fault simulator,” in *Proceedings of the 2008 17th Asian Test Symposium*, ser. ATS ’08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 35–40. [Online]. Available: http://dx.doi.org/10.1109/ATS.2008.17

[24] Lifting. [Online]. Available: http://gforge-lirmm.lirmm.fr/gf/project/lifting/