Enhancing the NekoStat Tool with Uncertainty, Resolution and Intrusiveness Evaluation Capabilities

Andrea Bondavalli [1], Andrea Ceccarelli [1], Lorenzo Falai [2], Michele Vadursi [3]
[1] Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, Viale Morgagni 65, I-50134 Firenze
Phone : +390554237457 – Email : bondavalli@dsi.unifi.it; andrea.ceccarelli1@alice.it
[2] ResilTech s.r.l. – Technologies for Resilience, Via Giuntini 63, I-56023 Navacchio (PI)
Phone : +390507519794 – Email : lorenzo.falai@resiltech.com
[3] Dipartimento per le Tecnologie, Università di Napoli "Parthenope", Centro Direzionale Isola C4, I-80143 Napoli
Phone : +390815476791 – Email : vadursi@uniparthenope.it

Abstract *– Dependability and resilience measurements pose some important challenges to the scientific and industrial community, which span from the proper definition of the measurand, to the adequate characterization of measurement tool and techniques, and the correct presentation of measurement results. The authors have recently investigated the general awareness about metrological issues by examining some of the most significant and popular tools and experiments of dependability and resilience measurements. Stemming from the results of this survey, the steps taken to enhance NekoStat, a tool for quantitative evaluations in distributed systems, are presented in the paper. Enhancements involve the capability of evaluating uncertainty, resolution and intrusiveness with regard to time interval measurements.*

Keywords *– Resilience Assessment, Measurements on distributed systems, Uncertainty, Resolution, Intrusiveness, Distributed clocks.*

I. INTRODUCTION

Measuring, assessing and benchmarking resilience is a hot topic nowadays for researchers as well as people working in the industry. The importance of such activity spans from verification and validation (V&V) to fault forecasting, from performance comparison to security assurance [1]. Some topical questions involve the definition of the most suitable metrics for resilience, the proper design and characterization of measuring systems and tools, the correct presentation of measurement results and the definition of common rules for a significant benchmarking.

The different approaches that can be followed to measure the resilience of a system or a component basically belong to three categories: analytic, simulative and experimental approaches. Which is the most adequate for quantitative assessment depends on the complexity of the system, the development stage of the system, the specific aspects to be studied, the attributes to be evaluated, the accuracy required, and the resources available for the study. Both analytic and simulative approaches are generally cheap for manufacturers and have proven to be useful and versatile in all the phases of the system life cycle. The accuracy of the results obtained through an analytic approach is strongly dependent on the accuracy of the values assigned to the model parameters and on how realistic the assumptions the system model is based on are. The simulative approach is one of the most commonly used approaches for quantitative evaluation in practice; as for the analytic approach, the accuracy of the obtained evaluation depends on the accuracy of the assumptions made for the system to be analyzed, as well as on the behavior of the simulation environment, and on the simulation parameters.

Although not always easy to realize, and generally more expensive than simulative and analytic approaches, experimental evaluation of resilience metrics and properties is becoming a more and more popular option. The opportunity of assessing a system in its real execution and its real usage environment is attractive and cannot be granted by analytic or simulative approaches. On the other hand, special care must be paid in the design and the execution of the measurements and in the presentation of the results, since there is still a gap between resilience measurements and the common practice and notions of metrology. Moreover, the approach followed to quantitatively assess algorithms and systems generally varies from a case to another, making the comparison among different results quite arduous, if not meaningless. This is basically due to the fact that there is still no significant interaction between people belonging to the dependability scientific community and experts in the field of instrumentation and measurement. In fact, measurement tools designed and utilized to evaluate dependability attributes of computer systems and algorithms are seldom recognized as measuring instruments.

Based on these premises, the authors, who separately belong to the dependability and the instrumentation and measurement communities, have recently started a discussion on possible cross-fertilization between these two research areas. First, they surveyed about 20 of the most important and significant tools and experiments addressed to measure dependability properties [2]. The scope was to investigate the general awareness about metrological issues in dependability and to understand if and to what extent systems and results were properly characterized and presented. The results show that some general awareness about some metrological issues is indeed present, but the followed approaches are quite intuitive, and usually quite incomplete, as well. What is notable, no quantitative information on the quality of measurement results is ever provided. Moreover, repeatability and resolution are rarely taken into consideration.

In the following, the natural evolution of such approach is presented. On the basis of the results of the survey, NekoStat [4], a software tool that allows quantitative evaluations of algorithms, has been enhanced in order to make it capable of evaluating measurement uncertainty, resolution and intrusiveness. The first results in this direction are presented in [27], where it is shown how the reliability of results obtained using NekoStat can be significantly enhanced thanks to uncertainty evaluation. Now, the tool has been equipped with uncertainty, resolution and intrusiveness evaluation capabilities.

The paper is organized as follows. The state of the art of metrology issues in dependability measurements is presented in Section II, which also explains the motivations of the paper. A brief description of NekoStat is provided in Section III, whereas the proposed enhancement is described in Section IV. Finally, conclusions are given in Section IV.

II. STATE OF THE ART AND MOTIVATIONS

The survey presented in [2] investigates if and to what extent the papers presenting some of the most relevant and popular tools and experiments for resilience measurements characterize the tools and present measurement results according to measurement theory [5]-[21]. The attention is focused on uncertainty, repeatability, intrusiveness and resolution [26], which appear to be the most significant metrological issues for resilience measurements.

Measurement uncertainty is a parameter characterizing the dispersion of the quantity values being attributed to a measurand. Uncertainty has to be included as part of the measurement result and represents an estimate of the degree of knowledge of the measurand. It has to be evaluated according to conventional procedures [23].

Resolution is the ability of a measuring system to resolve among different states of a measurand. It is the smallest variation of the measurand that can be meaningfully distinguished, i.e. that determines a perceptible variation of the instrument's indication.

Repeatability is the property of a measuring system to provide closely similar indications in the short period, for replicated measurements performed independently on the same measurand through the same measurement procedure, by the same operator, and in the same place and environmental conditions.

Intrusiveness indicates the perturbation that a measuring system induces on the measurand, determining a modification of its value.

The works considered in [2] cover some very different situations in which dependability measurements have been performed: from fault-injection tools and experiments (e.g. [6],[11],[17]), to general prototyping frameworks (e.g. [13]), from fail-aware systems [20] to experiments in which a total ordering protocol is tested [21]. The most significant results, which represent a state of the art, are summed up in the following.

As regards uncertainty, two cases deserve be cited: Loki [11],[12] and FORTRESS [20]. In the tool Loki, a post-runtime analysis, using off-line clock synchronization, is used to place injections on a single global timeline and to determine whether the intended faults were properly injected: there is a significant attempt to evaluate the uncertainty of the time instant at which faults were injected, even though it is not referred to as uncertainty [11], [12]. Such a deep analysis is performed only for timestamping fault injection. Although the approach to uncertainty is quite informal, far from uncertainty as dealt with by the Guide to the expression of Uncertainty in Measurement (GUM) [23], this example denotes a significant and remarkable interest in quantitatively evaluating the dispersion of the values that can be reasonably attributed to the measurand. Uncertainty-related issues are also taken into account only in the experiments related to the testing/development of FORTRESS [20]. FORTRESS is a support system for designing and implementing fault tolerant distributed real-time systems that use commercial off the shelf (COTS) components. In the test case performed, when measuring the one-way delay, the FORTRESS fail-aware datagram service computes an upper bound on the transmission delay of each delivered message.

A deeper awareness is present with regard to intrusiveness, which is not unexpected in computer science [5]-[8],[10],[11],[14],[15],[19]. Intrusiveness is addressed and analyzed at different levels of detail in [5],[8],[10],[14] and [19]. The tool Loki performs a post-runtime analysis, in order to be as non-intrusive as possible, rather than blocking the system at runtime while notifications about the system state are in transit [11]. The fault-injection tool ORCHESTRA, which deals with real-time systems with strict time requirements, goes further [14]. It is designed to explicitly address the intrusiveness of fault injection on a target distributed system. This operation is performed by exploiting the operating system support to quantitatively assess the intrusiveness of a fault injection experiment on the timing behaviour of the target system and to compensate for it whenever possible.

On the contrary, resolution is not considered at all, although it is usually easier to be evaluated than uncertainty and intrusiveness. Indeed, in some cases (e.g. when Java system calls are used to collect timestamps) it can be a limiting factor (e.g. [13]).

Finally, the difficulty in reaching a satisfactory level for repeatability has been taken into account in the experiments on computer systems described in [19], even if the word repeatability is not explicitly used. The authors show awareness that, due to the aforementioned limits on accurate timestamping, many executions of the same run will probably not bring exactly the same results. This – they say - explains why a second execution of the same run does not necessarily recreate a catastrophic incident that can, for instance, occur in the first execution. The problem of creating repeatable experiments is discussed also in [5] and [9]. Repeatability is probably the most critical issue to face, especially when distributed systems are considered.

It is worth highlighting that dependability and resilience measurements on computing systems involve a

wide variety of measures, from discrete measures, such as number of source code lines, number of system calls (which are typical software quality measurements), packet size in packet-switched networks, queue size of a particular network protocol, number of packets sent on top of a point-to-point channel, number of messages received by a network interface, maximum memory size of an application, maximum number of records in a database table, to continuous measures that refer to the dynamic behavior of the system under evaluation, which include delays experienced in an end-to-end connection, quality of clock synchronization, Mean Time To Failure (MTTF), Mean Time Between Failures (MTBF), and many other direct and indirect measures related to distributed events. The latter class includes critical measures that present more challenges than those belonging to the former one. A closer look at this class highlights the crucial role of time measurements. Dependability-related measurements are very often based on measuring timestamps and time intervals, either because the measurand is a time interval, or because the measurand is obtained through indirect measurements based on time. This is the reason why, in order to enhance NekoStat in the direction of a suitable measurement tool, the attention has been first focused on distributed time interval measurements, which appear to represent the majority of critical dependability and resilience measurements.

III. THE NEKOSTAT TOOL

Neko is a simple but powerful and highly-portable Java framework that allows to define and analyze distributed algorithms, showing the attracting feature that the same Neko-based implementation of an algorithm can be used for both simulations and experiments on a real network [13]. The architecture of Neko can be divided into three main layers: from the top layer to the bottom layer, there are applications, which are developed by the programmer/user, NekoProcesses, that are the core of the testing framework, and networks, that are network interfaces (real or simulated). Neko directly supports the assessment of qualitative properties of distributed systems, following a dynamic verification approach.

NekoStat is an analyzer, i.e. a software tool that allows quantitative evaluations of algorithms implemented using the Neko tool. In fact, NekoStat adds to Neko the ability to collect events and to analyze them by statistical and mathematical tools [4]. NekoStat is composed by two disjointed sets of functionalities: analysis functionalities, to manage and collect distributed events, and mathematical functionalities, to draw statistical analysis of the interesting metrics. Regarding execution on a real network, there is a distinction between the master process and the others processes, called slaves: when execution ends, all events collected by slaves are sent to the master, which makes an *a posteriori* analysis (off-line analysis). Figure 1 depicts a high level view of the complete architecture of Neko/NekoStat tool.

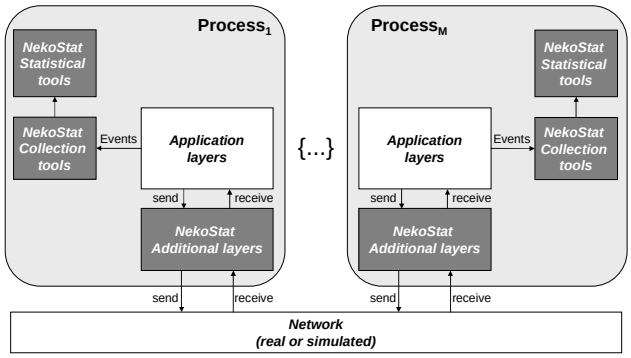

Fig.1. Architecture of Neko/NekoStat

IV. ENHANCING NEKOSTAT

According to the achievements found in [2], NekoStat provides no information on intrusiveness, uncertainty, resolution and repeatability, and does not explicitly address resolution. However, the main limitation of NekoStat as an instrument for resilience and QoS measurements on distributed systems is that it does not allow to evaluate the quality of the measurements it performs. As is, NekoStat cannot be qualified as a good measuring instrument. Apart from the formal metrological correctness, the practical risk is that the tool collects measurement results which are not reliable, without discarding them before the data processing.

Letting apart repeatability, since repeatable conditions are very difficult to achieve in measurements of distributed systems, the NekoStat tool has been equipped with the capability of evaluating measurement uncertainty, intrusiveness and its resolution with regard to distributed time interval measurements.

A. Uncertainty

Whenever we want to obtain direct or indirect measurements in which time intervals related to different nodes of a distributed system are involved, it is *important* and *useful* to evaluate the measurement uncertainty in order to be aware of the quality of measurement results. In these cases, the most significant cause of uncertainty is usually represented by poor synchronization between distributed clocks. When time intervals, such as one-way delay, are measured in a computer network, synchronization protocols such as Network Time Protocol (NTP) [24] are commonly exploited to limit the misalignment of distributed clocks due to both offsets and drifts among clocks. Nevertheless, the offset of a local clock from time reference is a hard-to-predict factor, which may vary due to many causes such as unexpected network delays, temperature variations or even faults in clock synchronization mechanisms. So, exploiting protocols like NTP does not totally prevent from collecting some severely incorrect data due to transient perturbations which cause an increase in the clock offset, and ultimately, an increase in the dispersion of the values that can reasonably be attributed to the measurand (e.g.

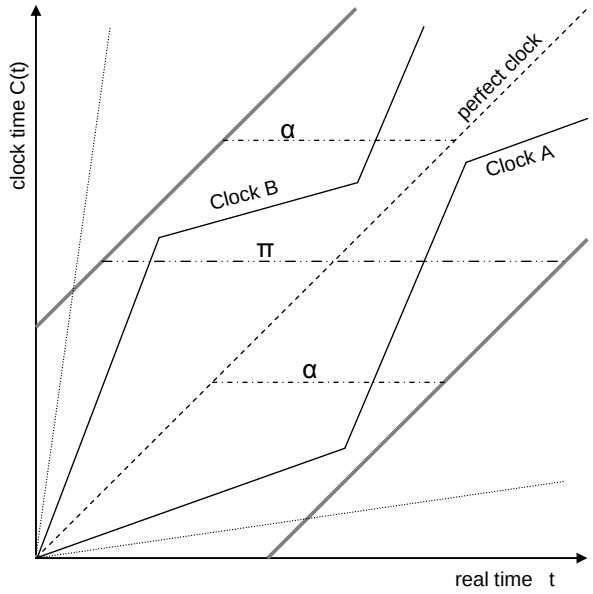

Fig.2. A set of two clocks: synchronization, accuracy, precision and drift.

time interval), which can be unacceptable for some applications. So, not only is uncertainty evaluation important to comply with rules and common practices in metrology [23], but it is also useful to prevent from trusting severely erroneous results.

To this aim, a new software component, Reliable and Self-Aware Clock (R&SAClock), has recently been developed. A preliminary analysis of R&SAClock is presented in [3]. R&SAClock, which is basically a process daemon, is capable of evaluating the synchronization of a set of distributed clocks. By integrating the R&SAClock into NekoStat, the analyzer can now obtain the timestamps it uses to measure time intervals in the form of *enriched timestamps* containing information on the quality of the actual clock synchronization, too.

Before presenting the integration of R&SA Clock with NekoStat, few notes on the terminology related to distributed clock synchronization are given. Let us consider a distributed system, composed of n processes P_1, P_2, ..., P_n; each of which has access to a local clock. The behavior of this local clock can be described defining three quantities, namely *precision*, *accuracy* and *drift*. Please note that the terms *precision* and *accuracy*, will be introduced in the following, with reference to their common meaning among researchers belonging to the dependability community [25]. These are clearly not coincident with their meaning in metrology [26].

Precision π describes how closely local clocks remain synchronized to each other at any time. Accuracy α_i describes how local clock of the process P_i is synchronized at any time to an absolute real time reference, provided externally; accuracy is thus an upper bound to the distance between local clock and real time. Accuracy makes sense only in presence of external synchronization, in which an external absolute real time reference is used as target of the synchronization. As a

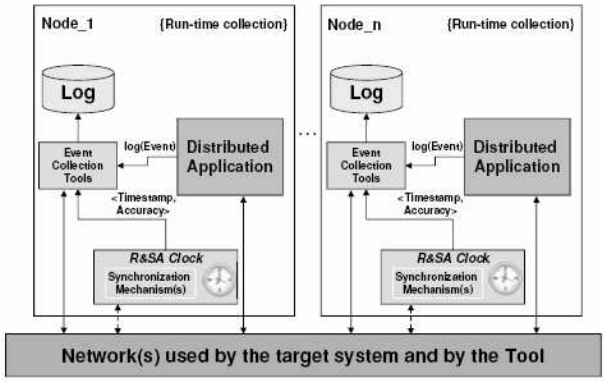

Fig.3. Instrumenting the system with R&SA Clock..

consequence of these definitions, considering for example a set of two clocks respectively with accuracy α_1 and α_2, precision is at least as good as $\pi = \alpha_1 + \alpha_2$. The drift describes the rate of deviation of a clock from a time reference. The clock synchronization can be defined as the process of maintaining the properties of precision, accuracy and drift of a clock set [25]. Figure 2 exemplifies the concepts of precision, accuracy, drift and clock synchronization. The outside thick dashed lines represent the bound in the rate of drift, a fundamental assumption for a clock synchronization, since it allows to predict the maximum deviation after a given time interval.

The main task executed by R&SAClock is providing information about current time and current time accuracy (as defined above), through monitoring the synchronization level of the local clock with respect to a global time reference. It interacts with synchronization mechanisms active on local machine and acts as an oracle of the quality of synchronization. Its services are available both for applications and middleware services.

The interface of R&SAClock towards applications running on the system consists of two simple primitives: *getTime*() and *setBound*(). The first one allows to get the enriched timestamp previously described, while the second one is used to set the *applicationAccuracy* bound, that will be described in the following.

When an application wants to know the current time and/or the current accuracy, it can use the procedure *getTime*() to query the R&SAClock. The primitive *getTime*() returns provides an enriched timestamp, constituted of four fields, <*time, minTime, maxTime, FLAG*>, where time is the time obtained simply reading the local clock, *minTime* and *maxTime* are respectively the left and right bounds of the reasonable values that can be attributed to time (in most of the cases the left and right bounds are symmetrical with respect to the parameter time; however, this depends on the characteristics of the adopted synchronization mechanism). Here, in the absence of more precise information and models to evaluate uncertainty, the relevant information that can be extracted is that the measurand is within the interval (*minTime, maxTime*). This way, uncertainty is not evaluated through a

statistical processing of repeated measurement results, but a *type-B* approach is followed, according to [23].

The generic application can inform NekoStat of the worst time accuracy that it can accept, by giving the parameter *applicationAccuracy* in input to R&SAClock. Thus, R&SAClock can compute its output FLAG, which is a Boolean value indicating whether the accuracy required by the application running on the system is within bound or not. Moreover, when several applications run on the same system and provide their *applicationAccuracy* values to the R&SAClock, the R&SAClock chooses the one closest to zero and tries to keep clock accuracy within the chosen accuracy bound, using the facilities of the synchronization mechanisms. However, it has to be noted that these facilities are available only on those synchronization mechanisms that allow local processes to command and force synchronization activities. As a further useful feature, in addition to evaluating the uncertainty of time interval measurements, NekoStat can also exploit *FLAG* to filter measurement results affected by unacceptably high uncertainty, and exclude them from the successive analysis.

A key parameter in R&SAClock is the value of local clock drift: in fact, the accuracy evaluation mechanism provided by the R&SAClock requires an estimation of a worst case bound for the clock drift [3]. At start-up, this parameter is read by R&SAClock from configuration files; this value can be further modified by the root user or by the R&SAClock itself, as a result of the analysis of local clock drift estimation and behavior over time.

In the version currently integrated in NekoStat (see Figure 3), R&SAClock has been implemented for NTP synchronization protocol and Linux operating system (OS). In this implementation, R&SAClock lays on NTP synchronization protocol and uses data and functionalities provided by NTP (via NTP-related system calls and NTP log files) to get current time from virtual system clock, and value necessary to set up the accuracy evaluation mechanism. Since NTP does not allow a user to force a synchronization, the ability to try to keep synchronized within an imposed bound is not currently implemented. This poses no limitation on the NekoStat capability of evaluating the quality of time synchronization, but simply means that the R&SAClock developed to interact with NTP acts like a completely passive monitor.

B. Intrusiveness

The enhancement with respect to intrusiveness evaluation consists of an offline analysis that the user can choose to perform before the beginning of an experiment. Once the tool has been configured for the experiment,, a specific workload is executed on the distributed system. A set of events is collected in each node, and the CPU time the tool uses to collect the event is evaluated. The *intrusiveness* of the tool is calculated as the *average value* of the time needed to collect a single event; it is an estimation of the intrusiveness of the run-time support used in NekoStat for the collection of the event. Notice that different nodes can have different value of intrusiveness, since different nodes can use different hardware and different OS; these values are exported by NekoStat and they can be used in the final evaluation (made off-line at the experiment termination).

C. Resolution

Regarding resolution of time interval measurements, it depends on the granularity of the virtual clock of the operating system, as well as on the software support available [3]. In distributed time interval measurements, the resolution is the maximum between the resolution of the source node and that of the destination node. Thus, NekoStat evaluates the resolution of each node before starting the experiments and then associates to each measurement result the corresponding resolution, as additional information to each measurement result.

V. CONCLUSION

The paper has presented a new approach to dependability and resilience measurements in distributed systems. The work is motivated by the results of a recent survey that the authors have conducted on the most relevant scientific literature in the field to verify if and to what extent metrological issues are born in mind by researchers in the dependability community. The results highlighted that there is some awareness about some relevant issues of metrology, but it is still too intuitive and far from the common practices in metrology. In particular, practically no attention is paid to quantitatively evaluating uncertainty, resolution, repeatability and intrusiveness. On the basis of these considerations, an enhanced version of NekoStat, an analyzer for distributed algorithms and systems, has been presented. The new version of the tool permits to evaluate uncertainty of time intervals measurements, as well as resolution and intrusiveness and is consequently more adherent to the rules and the common practices of metrology.

The ongoing research activity is focused on conducting a large experimental campaign to characterize NekoStat as a measuring instrument for a wider set of dependability and resilience measures.

REFERENCE

[1] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. on Dependable and Secure Computing*, vol.1, no.1, 2004.

[2] A. Bondavalli, A. Ceccarelli, L. Falai, M. Vadursi, “Foundations of measurement theory applied to the evaluation of dependability attributes,” *Proc. of 37th Annual IEEE/IFIP Intern. Conf. on Dependable Systems and Networks*, Edinburgh, Scotland, pp.522-533, 25-28 June 2007.

[3] A. Bondavalli, A. Ceccarelli, L. Falai, “A self-aware clock for pervasive computing systems,” *Proc. of 15th Euromicro Intern. Conf. on Parallel, Distributed and Network-Based Processing*, Washington, DC, USA, pp.403-411, 2007.

[4] L. Falai, A. Bondavalli, F. Di Giandomenico, “Quantitative evaluation of distributed algorithms using the neko framework: the nekostat extension,” *Proc. of Latin-American Symposium on Dependable Computing LADC 2005*, Salvador, Bahia, Brazil, pp.35-51, 25-28 October 2005.

[5] J. Carreira, H. Madeira, J. Silva. Xception, "Software fault injection and monitoring in processor functional units," *In Preprints 5th Int. Working Conf. on Dependable Computing for Critical Applications (DCCA-5)*, pp.135–49, 1995.

[6] J. Aidemark, J. Vinter, P. Folkesson, J. Karlsson, "Goofi: Generic object-oriented fault injection tool," *In Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)*, pp. 83–88, Washington, DC, USA, 2001.

[7] J.-C. Fabre, F. Salles, M. Rodriguez Moreno, and J. Arlat, "Assessment of COTS microkernels by fault injection," *In DCCA '99: Proceedings of the conference on Dependable Computing for Critical Applications*, page 25, Washington, DC, USA, 1999.

[8] M. Rodriguez, A. Albinet, J. Arlat, "Mafalda-rt: A tool for dependability assessment of real-time systems," *In DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 267–272, Washington, DC, USA, 2002.

[9] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, D. Powell, "Fault injection for dependability validation: A methodology and some applications," *IEEE Trans. Softw. Eng.*, 16(2):166–182, 1990

[10] T. K. Tsai, R. K. Iyer, "Ftape: A fault injection tool to measure fault tolerance," *In 10th AIAA Computing in Aerospace Conference*, San Antonio, TX; USA, 1995.

[11] R. Chandra, R. M. Lefever, M. Cukier, W. H. Sanders, "Loki: A state-driven fault injector for distributed systems," *In Proceedings of the 2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8)*, pp. 237–242, Washington, DC, USA, 2000.

[12] M. Cukier, R. Chandra, D. Henke, J. Pistole, W. H. Sanders, "Fault injection based on a partial view of the global state of a distributed system," *In Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems*, page 168, Washington, DC, USA, 1999.

[13] P. Urban, A. Schiper, X. Defago, "Neko: A single environment to simulate and prototype distributed algorithms," *In ICOIN '01: Proceedings of the 15th International Conference on Information Networking*, page 503, Washington, DC, USA, 2001.

[14] S. Dawson, F. Jahanian, T. Mitton, and Teck-Lee Tung. Testing of fault-tolerant and real-time distributed systems via protocol fault injection. *In FTCS '96: Proceedings of the The Twenty-Sixth Annual International Symposium on Fault Tolerant Computing (FTCS '96)*, page 404,Washington, DC, USA, 1996.

[15] S. Dawson F. Jahanian, "Probing and fault injection of protocol implementations," *In International Conference on Distributed Computing Systems*, pages 351–359, 1995.

[16] A. Kalakech, T. Jarboui, J. Arlat, Y. Crouzet, K. Kanoun. "Benchmarking operating system dependability: Windows 2000 as a case study," *In Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, pp. 261– 270, Washington, DC, USA, 2004.

[17] N. Ignat, B. Nicolescu, Y. Savaria, G. Nicolescu, "Softerror classification and impact analysis on real-time operating systems," *In Proceedings of the conference on Design, automation and test in Europe*, pages 182–187, 3001 Leuven, Belgium, 2006. European Design and Automation Association.

[18] P. Donzelli, M. Zelkowitz, V. Basili, D. Allard, K. N. Meyer. "Evaluating COTS component dependability in context," *IEEE Softw.*, 22(4):46–53, 2005.

[19] T. K. Tsai, R. K. Iyer, D. Jewitt, "An approach towards benchmarking of fault-tolerant commercial systems," *In Proceedings of the 26th Annual Intern. Symp. on Fault-Tolerant Computing (FTCS '96)*, page 314, Washington, DC, USA, 1996.

[20] C. Fetzer, F. Cristian, "Fortress: A system to support fail-aware real-time applications," *In IEEE Workshop on Middleware for Distributed Real-Time Systems and Services*, San Francisco, December 1997.

[21] M. Balakrishnan, K. Birman, A. Phanishayee, "Plato: Predictive latency-aware total ordering," *In Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*, pages 175–188, Washington, DC, USA, 2006.

[22] G. Bollella, B. Brosgol, J. Gosling, P. Dibble, S. Furr, M. Turnbull, *The Real-Time Specification for Java*, Addison Wesley, 2000.

[23] BIMP, IEC, IFCC, ISO, IUPAC, IUPAP, and OIML. *Guide to the expression of uncertainty in measurement*, 2nd edition, 1995.

[24] D. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. Communication*, vol.39, No.10, pp.1482-1493, 1991.

[25] P. Verissimo, L. Rodriguez, *Distributed Systems for System Architect*, Kluwer Academic Publishing, 2001.

[26] BIMP, IEC, IFCC, ISO, IUPAC, IUPAP, and OIML, *International vocabulary of basic and general terms in metrology*, 3rd edition, 2007.

[27] A. Bondavalli, A. Ceccarelli, L. Falai, M. Vadursi, "Towards Making NekoStat a Proper Measurement Tool for the Validation of Distributed Systems", *Proceedings of The 8th International Symposium on Autonomous Decentralized Systems*, Sedona, USA, pp. 377-386, 2007.