Measurement 80 (2016) 229–240

Contents lists available at ScienceDirect

Measurement

journal homepage: www.elsevier.com/locate/measurement

Differential analysis of Operating System indicators for anomaly detection in dependable systems: An experimental study

Andrea Bondavalli [a,1], Andrea Ceccarelli [a,1], Francesco Brancati [b,2], Diego Santoro [c,3], Michele Vadursi [c,*]

[a] Dipartimento di Matematica e Informatica, Università degli Studi di Firenze, Viale Morgagni 65, I-50134 Firenze, Italy
[b] ResilTech s.r.l. – Technologies for Resilience, Piazza Nilde Iotti 25, I-56025 Pontedera (PI), Italy
[c] Dipartimento di Ingegneria, Università degli Studi di Napoli "Parthenope", Centro Direzionale di Napoli Isola C4, I-80143 Napoli, Italy

ARTICLE INFO

Article history:
Received 21 May 2014
Received in revised form 19 October 2015
Accepted 3 November 2015
Available online 22 November 2015

Keywords:
System monitoring
Measurements on computer systems
OS anomalies
Anomaly detection
Dependability measurement
Fault detection

ABSTRACT

Dependable complex systems often operate under variable and non-stationary conditions, which requires efficient and extensive monitoring and error detection solutions. Among the many, the paper focuses on anomaly detection techniques, which monitor the evolution of some specific indicators through time to identify anomalies, i.e. deviations from the expected operational behavior. The timely identification of anomalies in dependable, fault tolerant systems allows to timely detect errors in the services and react appropriately. In this paper, we investigate the possibility to monitor the evolution of indicators through time using the random walk model on indicators belonging to Operating Systems, specifically in our study the Linux Red Hat EL5. The approach is based on the experimental evaluation of a large set of heterogeneous indicators, which are acquired under different operating conditions, both in terms of workload and faultload, on an air traffic management target system. The statistical analysis is based on a best-fitting approach aiming to minimize the integral distance between the empirical data distribution and some reference distributions. The outcomes of the analysis show that the idea of adopting a random walk model for the development of an anomaly detection monitor for critical systems that operates at Operating System level is promising. Moreover, standard distributions such as Laplace and Cauchy, rather than Normal, should be used for setting up the thresholds of the monitor. Further studies that involve a new application, a different Operating System and a new layer (an Application Server) will allow verifying the generalization of the approach to other fault tolerant systems, monitored layers and set of indicators.

© 2015 Elsevier Ltd. All rights reserved.

1. Introduction

It is well-known that dependable complex and real-time systems require to implement extensive monitoring functionalities to timely detect both hardware and software errors [1,21]. In the last years, several monitors and error detectors have been proposed to continuously check the health of the system being monitored and detect errors within specific time deadlines. Among the possible error

* Corresponding author. Tel.: +39 0815476791.
E-mail addresses: bondavalli@unifi.it (A. Bondavalli), andrea.ceccarelli@unifi.it (A. Ceccarelli), francesco.brancati@resiltech.com (F. Brancati), diego.santoro@uniparthenope.it (D. Santoro), michele.vadursi@uniparthenope.it (M. Vadursi).
[1] Tel.: +39 0554237457.
[2] Tel.: +39 0587212465.
[3] Tel.: +39 0815476791.

http://dx.doi.org/10.1016/j.measurement.2015.11.010
0263-2241/© 2015 Elsevier Ltd. All rights reserved.

detection approaches, attention has recently been focused on anomaly detection, which refers to the problem of finding patterns in data that do not conform to the expected behavior [6]. Such patterns are changes in the indicators characterizing the behavior of the system that are caused by specific and non-random factors, such as a system overload, or the activation of faults [3,5].

Anomalies can be detected using various techniques such as statistics, machine learning, model based and information theory. In general, a prior knowledge of the system behavior and/or a preliminary profiling phase are required, to define thresholds for the monitored indicators and rules used by the anomaly detector. This information can be gained by using a system model, either defined from experience or inferred from data collected on field [4]. For example, in [5] the CPU consumption of transactions in web application is modeled on the basis of statistical linear regression. The model is then used to detect performance anomalies, namely those changes in CPU usage that are not clearly justified by the actual workload. In [2] the authors propose a configurable detection framework to reveal anomalies in the Operating System (OS) behavior, related to system misbehaviors (software faults injected at the application level). For each monitored OS indicator, the framework computes lower and upper adaptive thresholds in order to take into account the dynamic behavior of the system. If the value of a specific indicator does not fall within the estimated interval, it is marked as suspicious. All suspicious indicators are combined to reveal an anomaly. The work in [2] models the behavior and dynamics of the OS indicators using the Gaussian random walk model [10,11], so that changes in the features of indicators caused by non-random factors are considered suspicious. The random walk model is intuitive and easy-to-model, thus resulting practical in many scenarios e.g., for the modeling of a software clock in [12–15]. While the results shown in [2] are encouraging, it has to be noted that the random walk model is empirically used without any quantitative assessment of its validity for the indicators taken into consideration, nor any motivated choice of the expected probability distribution.

The objective of the paper is to analyze the appropriateness of the random walk model to represent the behavior of the monitored variables belonging to the OS, through an experimental study. Specifically, the target system is Linux Red Hat EL5 and the approach is based on the experimental evaluation of 16 heterogeneous indicators, selected among the many offered by the OSs due to their semantics which bounds them to part of the OS that are mostly impacted by system anomalies, acquired under different operating conditions, both in terms of workload and faultload, on a target system. These monitors are implemented as a loadable kernel module by means of the SystemTap tool [16], which allows to program breakpoint handlers in a high level scripting language. As for practical applicability, this means that algorithms for the anomaly detection at OS level could be properly designed by analyzing the first-order time differences of some monitoring variables, represented by OS indicators: the analysis can lead to the definition of monitoring rules that can be applied in anomaly detectors built at the OS levels.

The insights of this work have potential to be verified (and thus generalized) for other OSs or different system layers (e.g., middleware software as the Application Servers), as well as different kinds of systems, e.g. for the anomaly detection monitoring in cloud computing infrastructures [26], or the anomaly-based resource usage monitoring in large cluster systems [27].

While it is well known from detection theory that a statistical analysis of the first-order time differences can be useful to detect faults [35], there is no established statistical model for the OS-level heterogeneous indicators considered in our experimental study. Indeed, to the best of the authors' knowledge, there has been no attempt to approach anomaly detection in dependable systems through a differential analysis of the system indicators.

The paper is the extended version of [20], which presented a simple qualitative analysis of the first-order differences of monitored indicators to evaluate whether the application of a Gaussian random walk approach was feasible for such systems. The value of this extended version of the paper lies in the quantitative assessment of the validity of the random walk approach for the heterogeneous indicators selected at OS level, and in the determination of possible alternative reference cumulative distributions functions (CDFs), different from the Normal, to fit the data. In particular, the experimental data are analyzed by comparing their goodness of fitness with the Gaussian, Cauchy, Laplace and Skellam theoretical distributions. The metric used to the scope is the Cramer–Von Mises integral distance. The results of the analysis show that for a large number of cases, the histogram of the first order time differences well approximates a Cauchy or a Laplace distribution, independently of the nature of the indicator and its statistical distribution. So, the conclusions given in [20] have to be partially corrected as a consequence of the statistical analysis presented here.

The rest of this paper describes basics on dependable systems and anomaly detection in Section 2, details on the experimental campaign in Section 3, experimental results in Section 4 and conclusions in Section 5.

2. Anomaly detection in dependable systems

This section summarizes the main concepts and definitions related to dependable computing and anomaly detection that are necessary for a complete understanding of the work; an exhaustive description of this set of concepts and definitions can be found in [1,21] for dependability and in [5] for anomaly detection in dependable systems.

2.1. Dependability concepts

A *service* is defined as a sequence of external states of the system, and a *service failure* (or simply, *failure*) as an event that occurs when the delivered service deviates from the correct service. *Dependability* of a system is the ability to avoid failures that are more frequent and more severe than acceptable. Deviation from correct service may assume different forms that are called *failure modes* and are ranked according to failure severities. An error is instead the part of the total state of the system that may lead to its subsequent service failure. The adjudged or hypothesized cause of an error is called a *fault*.

A fault is *active* when it produces an error; otherwise, it is *dormant*. *Error propagation* within a given component is caused by the computation process; a failure occurs when an error is propagated to the service interface and causes the service delivered by the system to deviate from correct service. Failure of a system causes a permanent or transient external fault for the other system(s) that receive service from the given system. This set of mechanisms constitutes the "*fault-error-failure*" *chain* of threats shown in Fig. 1.

Failures in complex systems have a variety of possible causes, which range from software to hardware, up to human errors, which are the most probable form of operational error [22]. It is not rare that failures "involve complex combinations of equipment failure, environmental factors, human error, and other causes" [23]. Among other possible causes of failure we have environmental causes, such as wide temperature variations, material aging and fatigue [22].

Means to attain dependability can be grouped into four categories: (i) *fault prevention* to prevent the occurrence or introduction of faults; it is typically part of general engineering activities; (ii) *fault tolerance*, whose introduction is facilitated by the addition of specialized support systems (e.g. for QoS monitoring, [24]), to avoid service failures in the presence of faults; (iii) *fault removal* to reduce the number and severity of faults, e.g. the fault tolerance mechanisms of the systems may be tested by fault injections tests [25]; (iv) *fault forecasting* to estimate the present number, the future incidence and the likely consequences of faults. The latter is conducted by performing a qualitative or quantitative evaluation of system behavior with respect to fault occurrence or activation.

Our work, like other anomaly detection approaches for dependable systems monitoring in general, can be considered as part of the *fault tolerance* group, as their aim is to collect indications, whenever not evidence, of the occurrence of faults and the consequent generation of errors in system's services.

2.2. Anomaly detection

Modern complex software systems usually consist of many interacting components and layers, including operating systems and network protocols, virtual machines, middleware technologies and OTS items that all together may reach millions of lines of code (and several connections and physical nodes). Revealing all software faults with pre-operational testing is very difficult and expensive and exhaustive testing is typically unfeasible. As a result, such systems suffer from residual faults, i.e., faults that escape testing and get activated only during operation.

Very often these software systems have a long lifetime, during which they evolve as they are integrated with other (possibly legacy) systems, and/or are updated either to reflect changes in their requirements or for bugs fixing. Such evolution is reflected in the progressive increase in the complexity of services and often causes services to operate beyond the design conditions that were initially planned for them. Also, new and unexpected overload conditions could occur as a consequence of a new and heavier workload, which can result in failures and costly service downtime.

The IEEE Standard 1044-1993 [31] defines an *anomaly* as "*any condition that deviates from expectations*". These expectations represent nominal/desired behavior that may be derived from requirements, design documents, standards and on-field experience.

Anomaly detection is an important mean to design monitoring procedures for timely detection of errors or attacks, since anomalies may be related to the activation of faults, performance issues, and malicious activities [29,30].

Engineers often use knowledge of the system behavior or preliminary profiling phases to define worst-case thresholds and *a-priori* rules to detect anomalies [4,28].

Recent studies show that revealing anomalies at OS level is a promising approach when traditional detection mechanisms (e.g., based on event logs, probes and heartbeats) exhibit poor performance or have limited applicability [33,34]. The driving idea is to shift the observation perspective to the OS, monitoring its behavior and interactions with the applications. The aim is to detect OS anomalies, such as system call errors and scheduling delays, which may be symptoms of incorrect behaviors. The approach is particularly suited for off-the-shelf (OTS) and legacy-based services, as it does not require to modify the service itself but only the underlying layer (the OS). Revealing anomalies at OS level is a general approach that can be applied to a variety of circumstances and applications in a much more efficient and cheap way than instrumenting the applications themselves. In fact, instead of re-instrumenting each application each time, it would only be necessary to tune an already existing instrumentation.

3. The experimental dataset

The dataset that has been used for the analyses is the result of the experimental campaign performed in [2], where the authors implemented an instrumentation infrastructures able to collect OS-level indicators, both for Linux and Windows environments. The testing activity was performed by analyzing a large amount of data monitored in a real and complex case application, namely the SWIM-BOX® [2], a prototype to support global interoperability for the novel Air Traffic Management (ATM) systems. SWIM-BOX is deployed on Windows and Linux platforms. It is a complex OTS (Off-The-Shelf)-based application, which offers several facilities to SWIM-BOX users: synchronous/asynchronous communication pattern (i.e. request/reply, publish/subscribe), security services (e.g., authentication, authorization, encryption) and distributed and transactional data storage. The application is made of several OTS, e.g., OS, the application Server (JBoss) and the data distribution middleware (OpenSplice).

The experiments consist in the execution of several functional and performance tests. In addition, faults are injected to mimic the activation of residual software faults in the data distribution middleware during operation. Errors

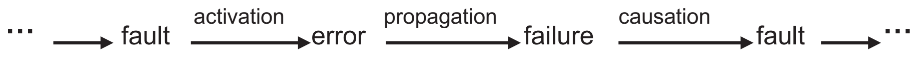

Fig. 1. Fault–error–failure chain.

resulting from the injection of software faults may propagate to the interface of the JBoss application server by leading to: (i) hang [8]; (ii) crash; or (iii) content failures.

Using SWIM-BOX under Linux OS, the collection of system indicators is accomplished by means of probes dynamically inserted into the kernel, without modifying and recompiling the OS or application source code. A probe consists of a breakpoint (i.e. a special CPU instruction that suspends the execution of the kernel instruction) and a handler routine, which is executed at the breakpoint to provide the desired information (e.g., input parameters or return values of called functions). These monitors are implemented as a loadable kernel module by means of the SystemTap tool [16], which allows to program breakpoint handlers in a high level scripting language.

Using SWIM-BOX under Windows OS, the monitoring infrastructure has been implemented using the Windows Reliability and Performance Monitor (WRPM) [17]. WRPM is a monitoring tool, available on both Windows desktop and Server releases, which provides several functionalities to: (i) monitor application and hardware performance in real time; (ii) track the performance impact of applications and services; (iii) generate alerts and reports; (iv) take actions when user-defined thresholds are exceeded.

The OS level indicators that have been monitored in both systems are the following (in brackets we present an acronym that will be used in the following of the paper):

- system call errors (SYSCALL): number of error codes returned from system calls invocation;
- OS signals (SIGNAL): number of signals used for coordination or information purposes (e.g., invalid memory access, process crash, loss of a socket connection, I/O data available);
- task scheduling timeouts (TASK_SCHED_TO): number of timeouts expired since a process released the CPU;
- waiting time for critical section timeouts (MUTEX): number of timeouts expired since a process (thread) started waiting for entering a critical section;
- holding time in critical section timeouts (SCHED_THRESHOLD): number of timeouts expired since a process (thread) entered a critical section;
- process (thread) activation/termination (PROCESS_EXIT): number of processes (threads) starting/terminating their execution;
- disk Read/Write timeouts (DISK/ReadTO and DISK/WriteTO respectively for timeouts on reading and writing): number of timeouts expired since the last Read/Write operation on disks;
- socket I/O timeouts (NET/IN and NET/OUT respectively for inputs and outputs): number of timeouts expired since the last Read/Write operation on a network socket;
- disk I/O throughput (DISK/IN and DISK/OUT respectively for inputs and outputs): bytes transferred per time unit in operations on disks;
- network I/O throughput (UDP_SEND and UDP_RECV respectively for UDP inputs and outputs; TCP_SEND and TCP_RECV respectively for TCP inputs and outputs): bytes transferred per time unit in operations on network devices.

This experimental data are organized in an OLAP [7] data warehouse modeled through fact and dimension tables.

Facts tables contain an entry for each sample measured in experiments. An entry contains the values measured for the monitored resources and parameters of that particular experimental setup, namely the values of dimensions.

Dimension tables refer to specific characteristics of the experimental setup and they store the possible values for them. We consider the following dimensions:

- *Target System (SUT)*. Characteristics of the Target System, e.g., OS, number of CPU, CPU speed, RAM, disk speed. By SUT 1 we indicate a Linux testbed, equipped with Intel Xeon 2.5 GHz (4 cores) CPU, 8 GB RAM, running Red Hat Enterprise Linux 5.By SUT2 we indicate a Windows testbed composed by an Intel Pentium 4 3.4 GHz (2 cores), with 3 GB RAM, running Windows Server 2008.
- *Events*. Monitored events, which are described in this section.
- *Workload*. Characteristics of the adopted workload. Message rate, message burst rate (a burst is a collection of message received into a single interaction) and messages per burst have been varied through different executions.
- *Faultload*. The type of faults injected and the software component target of the injection. In this campaign we used code mutation technique.
- *Run*. Information on the executed run (e.g., start time, end time, sample period).
- *Scenario*. The description of the application scenario, e.g., the number of entities involved in the communication and their role (i.e. Manager Contributor).

4. Experimental results

While some of the system indicators monitored during the experimental campaign can be generally associated to discrete-time and continuous amplitude random processes, in many cases the system indicators take values in the set of natural numbers N (number of last scheduling timeout, number of allocated threads, number of disk read timeout, number of disk write timeout).

Figs. 2–4 show measurement results related to three different indicators, in absence of faults. They are related respectively to disk write timeout; net in throughput (Byte/s); and I/O write throughput (Byte/s). In each figure, the evolution versus time of the indicator is shown

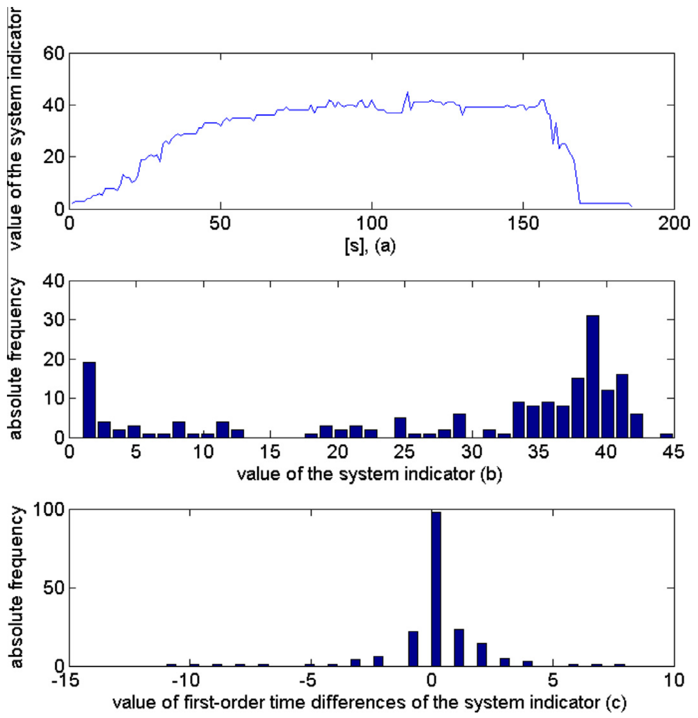

Fig. 2. SUT 1, workload 1, event 10 (disk write timeout), no faultload, run 8. (a) Signal, (b) histogram of the signal and (c) histogram of first order time difference.

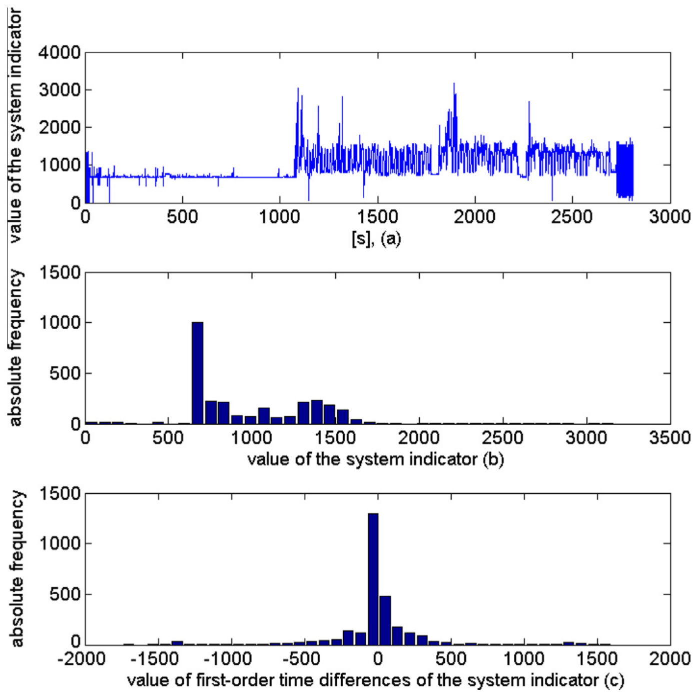

Fig. 3. SUT 2, workload 6, event 19 (net in throughput), no faultload, run 63. (a) Signal, (b) histogram of the signal and (c) histogram of first order time difference.

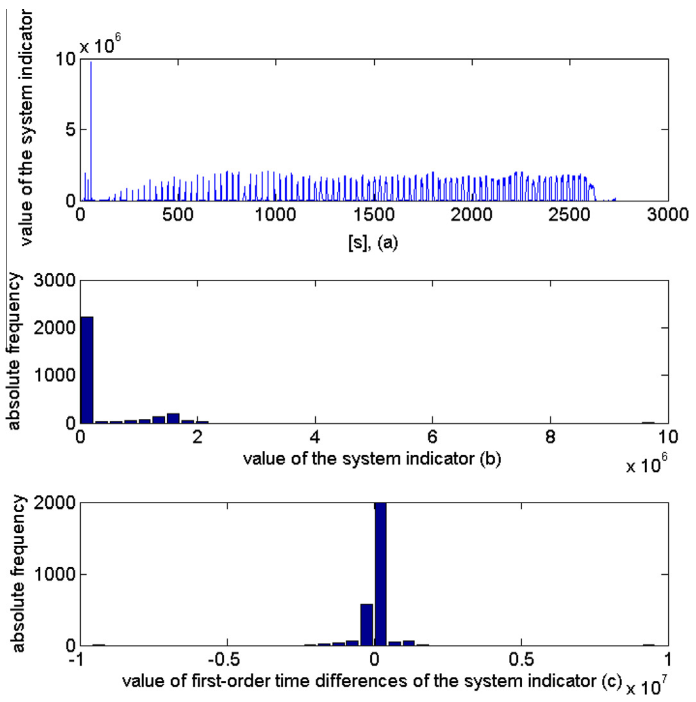

Fig. 4. SUT 2, workload 6, event 28 (IO write throughput), no faultload, run 63. (a) Signal, (b) histogram of the signal and (c) histogram of first order time difference.

(subfigure a), along with the histogram of the values acquired for the indicator (subfigure b) and the histogram of the first-order time differences of the indicator (subfigure c). The first order time difference Δ of the generic indicator x is defined as

$$\Delta(k) = x(k+1) - x(k) \tag{1}$$

where k is the discrete-time variable. Indicators values are acquired with a sampling period $T_c = 1$ s. The number of samples for each indicator varies with the experiment, ranging from few hundreds to few thousands.

Histogram analysis is often used to compare empirical distributions with theoretical ones and to estimate first and second order moments of random variables from measurement data [6,9]. As expected, the histograms in Figs. 2b, 3b and 4b show that the monitored system indicators have their "own" probability distributions, which in principle can depend on the specific run of the experiment, as well as on the characteristics of the machine under test and the workload, not to mention possible faults. In fact, changes in system workload can be considered as variable environmental conditions, in terms of the randomness they can induce on measurement results [32]. What is really interesting is that the histograms of the first order time differences Δ have very similar shapes, no matter what the original distribution of the indicators is. Indeed, even histograms of indicator values that appear very different in shape, range and characteristics give place to a related histogram of first order time differences that appears to be Gaussian-like (see Figs. 2c, 3c and 4c).

Similar outcomes have been observed in experiments with faults. As an example, Fig. 5 is related to the indicator "scheduling timeouts" (i.e., number of timeouts expired since a process released the CPU). The figure shows the histograms for the indicator and its first time difference, in presence of a natural fault (i.e., an error has been detected with no injection) starting after about 160 s. Fig. 5c shows that the histogram of the monitored system indicator first order time difference resembles a discretized Gaussian distribution, again. Another experiment with fault related to the indicator disk write timeout is shown in Fig. 6.

In both these cases, an inspection of subfigures c (Figs. 5c and 6c) suggests that the step size at the time of the fault is well over six-sigma limits (Fig. 5c, in a neighborhood of −216 and Fig. 6c, in a neighborhood of −18), so the system indicators can easily be considered out of control.

However, if we look back at Fig. 3, some observations of the first order time difference are well over 1000, in absolute terms, which could lead to false positive, if the indicator is used alone to detect a fault. It is thus advisable, and in line with the literature, to monitor several indicators jointly in order to establish if the whole system undergoes a fault.

4.1. Best-fitting approach

In order to obtain a model of the first-order time difference of the monitored indicators, the empirical cumulative distribution function (ECDF) of each monitored indicator, given by:

$$\mathrm{ECDF}(x) = \frac{\text{number of the elements in the sample} \leqslant x}{N} \tag{2}$$

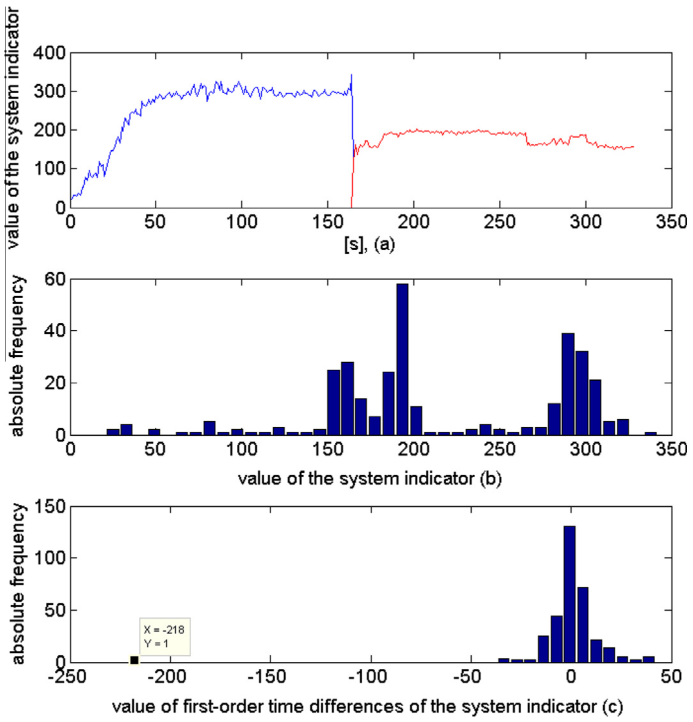

Fig. 5. SUT 1, workload 2, event 8 (scheduling timeouts), faultload 2, run19. (a) Signal, (b) histogram of the signal and (c) histogram of first order time difference.

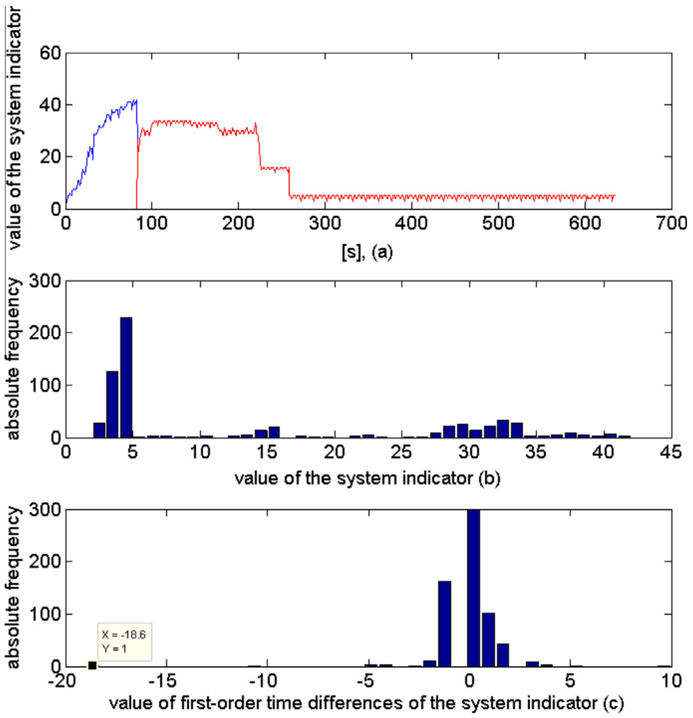

Fig. 6. SUT 1, workload 2, event 10 (disk write timeout), faultload 2, run 27. (a) Signal, (b) histogram of the signal and (c) histogram of first order time difference.

where N is the total number of the elements, has been compared to some models CDFs.

The model distributions have been selected according to the features of the histograms shown in the previous section. In particular, for variables representing throughputs (such as DISK/IN-OUT, UDP_SEND-RECV and TCP_SEND-RECV), which can be reasonably modeled as continuous-amplitude variables, models CDFs like the Laplace, the Cauchy and the Normal have been considered [18].

For the first-order time difference of discrete amplitude variables, such as PROCESS_EXIT, MUTEX and SYSCALL, the Skellam distribution was originally considered for fitting purposes. The reason for such a choice is the nature of the indicators. For example, let us consider the variable PROCESS_EXIT: its value at a given discrete-time instant k represents the number of processes (threads) that have started their execution in the time interval $(k\,T_c, (k+1)\,T_c)$. So, if the number of process that have started their execution at a given time t since the beginning of the experiment can be reasonably modeled as a Poisson process $\mathcal{N}(t, \lambda)$, the variable PROCESS_EXIT representing the difference $\mathcal{N}(k\,T_c, \lambda) - \mathcal{N}((k+1)\,T_c, \lambda)$, can be modeled as $\mathcal{N}(T_c, \lambda)$ for any k. Consequently, the first order time difference can be considered as the difference of two independent Poisson processes, i.e. a Skellam distribution of parameters $\lambda_1 = \lambda_2$ for $t = T_c$. As the experimental results show, however, the Skellam model does not come out to be suitable for fitting the empirical CDF. This is the reason why we have included also Laplace, Cauchy and Normal model CDFs in the tests with discrete amplitude variables. In fact, the paper is not focused on finding the optimal detector or the rigorous model for such heterogeneous indicators, but rather on refining the empirical approach used in [2], according to a quantitative assessment that is based on real measurements.

The selected model distributions are briefly reviewed here in the following.

The Laplace CDF with parameters μ and $b > 0$ is given by

$$F_X(x) = \begin{cases} \frac{1}{2} exp\left(\frac{x-\mu}{b}\right), & x < \mu \\ 1 - \frac{1}{2} exp\left(-\frac{x-\mu}{b}\right), & x \geqslant \mu \end{cases} \tag{3}$$

with $E[X] = \mu$, where $E[\cdot]$ is the statistical expectation operator, and variance $\mathrm{Var}[X] = 2b^2$.

The Cauchy CDF with parameters x_0 and y_0 is given by

$$F_X(x) = \frac{1}{\pi} \arctan \frac{x - x_0}{y_0} + \frac{1}{2} \tag{4}$$

where x_0 is the location of the peak of the probability density function, and y_0 is the scale parameter which specifies the half-width at half-maximum. The Cauchy distribution's mean and variance are undefined.

The CDF of the well-known normal distribution is

$$F_X(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x-\mu}{\sqrt{2\sigma^2}}\right)\right] \tag{5}$$

where $E[X] = \mu$, $\mathrm{Var}[X] = \sigma^2$, and $\mathrm{erf}(x)$ is the error function, defined as

$$\mathrm{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2}\, dt \tag{6}$$

The Skellam distribution is the discrete probability distribution of the difference of two statistically independent Poisson variables, with parameters λ_1 and λ_2. Its probability density function is

$$f_x(n) = e^{-(\lambda_1+\lambda_2)} \left(\frac{\lambda_1}{\lambda_2}\right)^{\frac{n}{2}} I_{|n|}\left(2\sqrt{\lambda_1 \lambda_2}\right) \tag{7}$$

where $I_n(x)$ is the modified Bessel function of the first kind, and being n integer the relation $I_n(x) = I_{|n|}(x)$ holds. In our cases, we have $\lambda_1 = \lambda_2 = \lambda$ and so

$$f_x(n) = e^{-2\lambda} I_{|n|}(2\lambda) \tag{8}$$

The criterion used to quantitatively assess the goodness of fit of the empirical distribution with the theoretical ones consists in the evaluation of an integral distance between the two CDFs. So, the model that best fits empirical data is the one whose integral distance from the ECDF is the smallest. The distance is defined according to Cramer–Von Mises (CV) [19]. In details, named $X_1, X_2, \ldots, X_N$ the observations of the variable that is to model, the CV distance d_{CV} between the continuous theoretical distribution $F_X(x)$ and the empirical distribution $F_X^*(x)$ is defined as

Table 1
Parameters and CV distance of the CDF fitting the experimental data.

Indicator	d_{CV}				Best fitting distribution	Parameters values
	Laplace	Normal	Cauchy	Skellam		
MUTEX	6.15	5.89	0.03	7.84	Cauchy (100%)	$x = -0.58$; $y = 0.05$
SYSCALL	1.25	2.03	0.75	2.69	Cauchy (100%)	$x = 1.65$; $y = 125.5$
SIGNAL	1.76	1.91	0.49	3.45	Cauchy (100%)	$x = -0.43$; $y = 3.37$
Task_Sched_TO	0.29	0.31	0.35	1.11	Laplace/Normal (50%/50%)	$\mu = 0$; $b = 8.94$ $\mu = -0.40$; $\sigma = 6.10$
Process_Exit	0.57	0.86	0.48	0.99	Cauchy (87.5%)	$x = -0.52$; $y = 4.51$
Sched_Threshold	0.62	1.74	0.39	1.84	Cauchy (75%)	$x = 0.59$; $y = 7.15$
DISK/ReadTO	1.81	2.36	3.56	4.95	Laplace (100%)	$x = -0.49$; $y = 0.37$
DISK/WriteTO	1.64	2.41	2.85	4.45	Laplace (100%)	$\mu = -0.42$; $b = 0.51$
NET/IN	0.61	1.39	0.43	7.30	Cauchy (71.4%)	$x = -3174$; $y = 302932$
NET/OUT	3.04	3.51	0.70	7.68	Cauchy (100%)	$x = -35.65$; $y = 2251$
DISK/IN	2.28	2.88	1.08	–	Cauchy (100%)	$x = -2437$; $y = 70520$
DISK/OUT	2.72	3.30	1.23	–	Cauchy (100%)	$x = -2092$; $y = 21844$
UDP_SEND	0.83	1.77	1.08	–	Laplace (87.5%)	$\mu = 0$; $b = 3.15$
UDP_RECV	0.58	1.32	0.99	–	Laplace (87.5%)	$\mu = 0$; $b = 2.96$
TCP_SEND	0.69	1.35	0.91	–	Laplace (62.5%)	$\mu = 0$; $b = 3.68$
TCP_RECV	0.79	1.38	0.96	–	Laplace (62.5%)	$\mu = 0$; $b = 4.18$

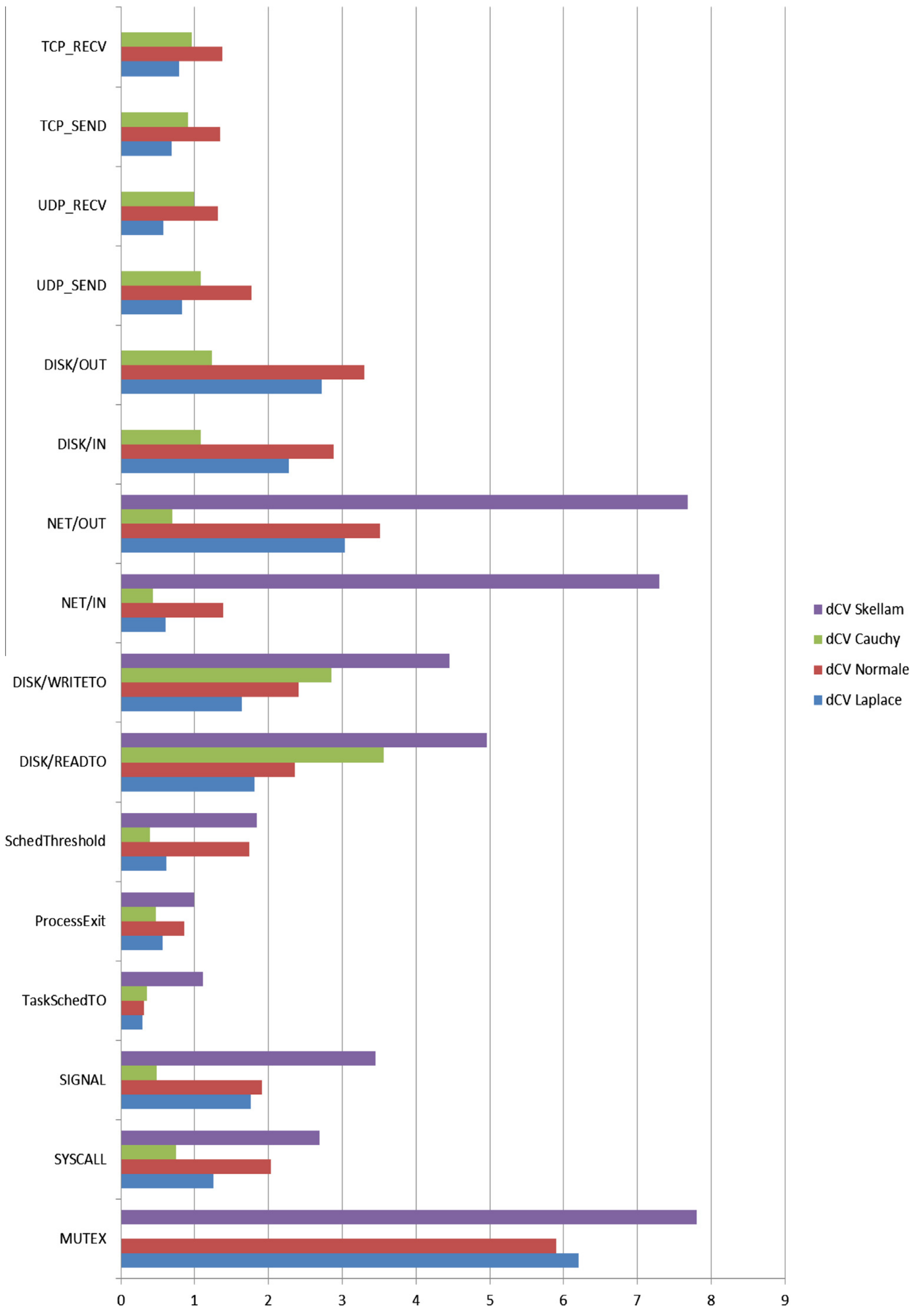

Fig. 7. Graphical representation of the CV distances in Table 1.

$$d_{CV}^2 = N \int_{-\infty}^{\infty} |F_X(x) - F_X^*(x)|^2 dF_X(x) \tag{9}$$

The integral in (9) can be calculated as [19]

$$d_{CV}^2 = \frac{1}{12N} + \sum_{i=1}^{N} \left| F_X(X_{(i)}) - \frac{2i-1}{2N} \right|^2 \tag{10}$$

where $X_{(i)}$ is the i-th order statistic from the set of observations. The parameters of the fitting CDFs have been chosen according to the moment matching procedure: the parameters μ and b of the Laplace distribution, the parameters μ and σ of the Normal distribution as well as the parameter λ of the Skellam distribution have been set so to ensure that the theoretical CDF has the same first and second order

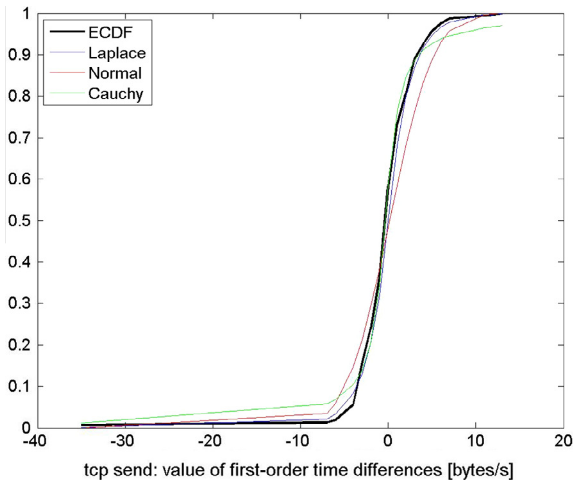

Fig. 8. Comparison of the ECDF and model CDF for the first-order time difference of the indicator TCP_SEND.

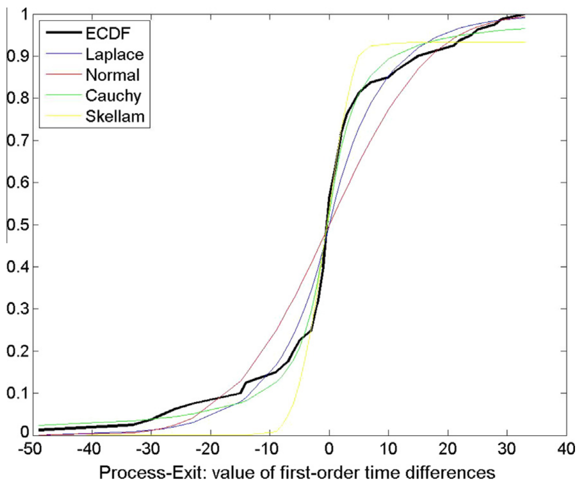

Fig. 9. Comparison of the ECDF and model CDF for the first-order time difference of the indicator Process_Exit.

moments of the ECDF. As the Cauchy distribution has undefined moments, its parameters x_0 and y_0 have been determined as the values that minimize the CV distance between the ECDF and the Cauchy CDF.

4.2. Best-fitting results

Table 1 reports the average values of CV distance measured for each continuous system indicator, with reference to the three considered model probability distributions. For the sake of clarity, the values in Table 1 are graphically represented in Fig. 7. Table 1 also indicates the reference distribution that best fits the experimental data, along with the parameters estimated for each distribution in the best fitting conditions. It is worth noting that the values of CV distances reported in Table 1 are average values over several runs, and the percentage values reported in brackets in the column "*Best Fitting Distribution*" represent the fraction of runs for which the related model distribution best fits the empirical data.

First of all, one can note that some indicators do not have a really good fitting with any of the distributions considered for comparison. These are DISK/ReadTO and DISK/WriteTO, which exhibit an average CV distance above 1.5. Similarly, an average CV distance greater than 1 is observed with regard to the indicators DISK/IN and DISK/OUT, which consequently cannot be said to have a good fitting with any of the considered standard CDFs.

On the contrary, a good fitting has been observed for some indicators. These are: Signal, NET/IN, Process_Exit, Sched_Threshold, MUTEX and Task_Sched_TO, all of which exhibit an average CV distance lower than 0.5. The first five of them best fit a Cauchy distribution, whereas the latter best fit a Laplace distribution.

Some conclusive observations are related to the groups of indicators for the timeout and the throughput. All the indicators that are related to timeout, such as the task scheduling timeouts TASK_SCHED_TO, the waiting time for critical section timeouts MUTEX, and the disk Read/Write timeouts DISK/ReadTO and DISK/WriteTO, show a preference for the Laplace distribution, even though the

Table 2
CV distance between considered Normal, Laplace and Cauchy distributions.

Indicator	d_{CV} Normal–Laplace	d_{CV} Normal–Cauchy	d_{CV} Laplace–Cauchy	d_{CV} Normal–Skellam	d_{CV} Laplace–Skellam	d_{CV} Cauchy–Skellam
MUTEX	0.03	0.39	0.96	0.28	0.26	0.17
SYSCALL	0.11	0.18	0.10	0.27	0.27	0.26
SIGNAL	0.19	0.34	0.32	0.37	0.50	0.52
TaskSchedTO	0.03	0.04	0.02	0.14	0.11	0.11
ProcessExit	0.06	0.10	0.05	0.16	0.12	0.10
SchedThreshold	0.25	0.25	0.05	0.15	0.17	0.15
DISK/READTo	0.19	0.24	0.23	0.22	0.25	0.28
DISK/WRITETO	0.25	0.28	0.16	0.21	0.27	0.28
NET/IN	0.09	0.14	0.11	0.46	0.46	1.02
NET/OUT	0.36	0.48	0.33	0.65	0.95	0.59
DISK/IN	0.15	0.27	0.22	–	–	–
DISK/OUT	0.19	0.29	0.26	–	–	–
UDP_SEND	0.19	0.20	0.07	–	–	–
UDP_RECV	0.13	0.14	0.06	–	–	–
TCP_SEND	0.16	0.16	0.05	–	–	–
TCP_RECV	0.17	0.18	0.05	–	–	–

CV distance is so high that we cannot say the Laplace distribution fits the experimental data well, in absolute terms. On the contrary, in the case of NET/IN and NET/OUT, the best fitting distribution is Cauchy, which fits the data well, with average CV distance equal to 0.43 and 0.70, respectively.

Similar considerations can be made for the indicators that are related to network I/O throughput (UDP_SEND, UDP_RECV, TCP_SEND and TCP_RECV). While the average CV distance is generally lower than 1 for both Cauchy and Laplace distribution, the latter exhibits the lowest d_{CV} and is therefore the best fitting model for such indicators. Figs. 8 and 9 show the ECDF of the first-order time differences of indicators TCP_SEND and Process_Exit respectively, with the theoretical CDF models whose parameters are estimated according to the procedure outlined above. The curves give evidence that the best fitting distributions are Laplace and Cauchy, respectively.

It is interesting to observe that the discrete variable Skellam gives good fitting results only for the two indicators Process_Exit and Task_Sched_TO. Nevertheless, in both cases, an alternative continuous variable distribution fits better the empirical data.

Finally, Table 2 reports the average values of the measured CV distance between couples of standard variables. Distances are generally low, which explains why rather good results were obtained in [2] where the hypothesis of Normal random walk was made without empirical evidence of its validity.

5. Conclusions

The paper has presented the results of an experimental study aimed at verifying the appropriateness of an anomaly detection approach based on random walk model for the observation of complex dependable systems through the monitoring of heterogeneous indicators at OS level. Specifically, we considered the Operating System Linux Red Hat EL5 and monitored 16 indicators, which were selected because considered the most appropriate to reflect the behavior of the system with regard to the occurrence of faults. The objective is to arrive to define monitoring rules that can be applied in anomaly detectors built at the OS levels.

As expected, for the system under test, the values assumed by different monitored indicators follow their own probability distribution, which is not known *a priori* and, in general, varies with the indicator taken into account. In spite of such heterogeneity in the indicators distributions, the paper has shown that the histograms of the first order time differences of such indicators are better approximated by a Cauchy and/or Laplace distribution in most of the cases, instead of a Gaussian distribution, as erroneously proposed in [20] on the basis of a purely qualitative analysis.

Being able to understand the behavior of key Operating System (OS) indicators through time allows building a monitoring system, and especially anomaly-detector systems as the one in [2], that can accurately distinguish anomalies by identifying the deviation of the actual behavior of the indicator from its behavioral model. In particular, by singling out indicators that can be statistically modelled with good approximation and discarding those which cannot be reduced to some model (i.e., indicators whose behavioral model is unknown or unpredictable, at least up to the current state of the art in complex systems monitoring), a two-fold benefit can be achieved: (i) the set of variables to monitor and ultimately the monitoring effort and overload introduced on the system can be reduced, and (ii) the significance of the analysis can be increased, as the monitoring thresholds can be fixed on the basis of the statistical analysis of empirical data under no faultload.

In line with such considerations, future work will consist in comparing the outcome of the traces processed in this work with traces logged in tests where faults are injected in the application, i.e. where anomalies can be identified through the monitoring of the OS indicators. The final goal is to design and implement a real-time anomaly detector based on monitoring the first order differences of a set of selected OS indicators.

We remark that at present there has been no attempt to approach anomaly detection in dependable systems through a differential analysis of the system indicators. The insights of this work that is focused on the Operating System Linux Red Hat EL5, have the potential to be verified and thus generalized for other OS or different system layers (e.g., middleware software as the Application Servers), as well as different kind of systems. Relevant examples are anomaly detection monitoring in cloud computing infrastructure in [27] and anomaly-based resource usage monitoring in large cluster system in [28]. This can potentially result in more accurate rules for modeling the nominal behavior of dependable system where anomaly-detector monitors are intended.

We aim to verify the generalization of the approach reproducing the experiments on systems with different settings and components. A testbed has been recently developed and described in [36] where the application is a customized version of the Liferay portal 6.1.2 Community Edition running on Tomcat 7.0.40 Application Server, and the Operating System is Linux CentOS 6 with kernel 2.6.32. This setting, together with an injector that introduces software errors in the code of the application and guides the tests [35], will allow to reproduce the experiments described in this paper. Additionally a new layer, the Application Server, is considered and its indicators are observed to detect anomalies [35,36]. This made necessary to identify a new set of indicators that describe the behavior of an Application Server, in a similar way as we progressed for the Operating Systems.

Acknowledgements

This work was partially supported by the European Commission through the FP7-2012-324334-CECRIS (CErtification of CRItical Systems), by Regione Toscana-Italy through the “Secure!” Research project (POR-CREO 2007-2013) and by the Italian Ministry of Education, University and Research the PRIN Project “TENACE-Protecting National Critical Infrastructures from Cyber Threats” (n. 20103P34XC).

References

[1] A. Bondavalli, A. Ceccarelli, L. Falai, M. Vadursi, A new approach and a related tool for dependability measurements on distributed systems, IEEE Trans. Instr. Meas. 59 (4) (April 2010) 820–831.
[2] A. Bovenzi, S. Russo, F. Brancati, A. Bondavalli, Towards identifying OS-level anomalies to detect application software failures, in: Proc. of IEEE International Workshop on Measurements and Networking (M&N), Anacapri, Italy, 10–11 October 2011, pp. 71–76.
[3] N. Delgado, A.Q. Gates, S. Roach, A taxonomy and catalog of runtime software-fault monitoring tools, IEEE Trans. Softw. Eng. 30 (12) (2004).
[4] G. Khanna, P. Varadharajan, S. Bagchi, Automated online monitoring of distributed applications through external monitors, IEEE Trans. Depend. Secur. Comput. 3 (2) (2006) 115–129.
[5] V. Chandola, B. Arindam, K. Vipin, Anomaly detection: a survey, ACM Comput. Surv. (CSUR) 41 (3) (2009) 15.
[6] D.C. Montgomery, Statistical Quality Control, McGraw-Hill, 2000.
[7] H. Madeira, J. Costa, M. Vieira, The OLAP and data warehousing approaches for analysis and sharing of results from dependability evaluation experiments, in: International Conference on Dependable Systems and Networks, 22–25 June 2003, pp. 86–91.
[8] G. Carrozza, M. Cinque, D. Cotroneo, R. Natella, Operating system support to detect application hangs, in: International Workshop on Verification and Evaluation of Computer and Communication Systems, VECoS, 2008
[9] P. Melillo, D. Santoro, M. Vadursi, Detection and compensation of interchannel time offsets in indirect fetal ECG Sensing, IEEE Sens. J. 14 (7) (2014) 2327–2334.
[10] P. Révész, Random Walk in Random and Non Random Environments, second ed., World Scientific Publishing Company, 2005.
[11] M. Picardello, W. Woess, Random Walks and Discrete Potential Theory, Cambridge University Press, 1999.
[12] A. Bondavalli, F. Brancati, A. Ceccarelli. Safe estimation of time uncertainty of local clocks, in: Proc. of Int. IEEE Symp. on Precision Clock Synch. for Measur., Contr. and Comm., ISPCS, 2009, pp. 47–52.
[13] P. Ferrari, A. Flammini, S. Rinaldi, A. Bondavalli, F. Brancati, Improving robustness of the synchronization quality of IEEE1588 nodes, in: Proc. of International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication, ISPCS, 2010, pp. 36–41.
[14] A. Bondavalli, F. Brancati, A. Flammini, S. Rinaldi, Master failure detection protocol in internal synchronization environment, IEEE Trans. Instrum. Meas. 62 (1) (2013) 4–12.
[15] A. Bondavalli, A. Ceccarelli, L. Falai, M. Vadursi, Resilient estimation of synchronization uncertainty through software clocks, Int. J. Crit. Comput.-Based Syst. 4 (2013).
[16] B. Jacob, P. Larson, B. Leitao, S.A.M.M. Da Silva, SystemTap: instrumenting the Linux kernel for analyzing performance and functional problems, IBM Redbook REDP-4469-00, January 2009.
[17] http://technet.microsoft.com/enus/library/dd744567(WS.10).aspx.
[18] M. Evans, N. Hastings, B. Peacock, Statistical Distributions, Wiley Interscience, 1993.
[19] R.B. D'Agostino, M.A. Stephens, Goodness of Fit Techniques, M.A., New York, 1986.
[20] A. Bondavalli, F. Brancati, A. Ceccarelli, D. Santoro, M. Vadursi, Experimental analysis of the first order time difference of indicators used in the monitoring of complex systems, in: Proc. of IEEE International Workshop on Measurements & Networking, M&N2013, Naples, Italy, 7–8 October 2013, pp. 138–142.
[21] A. Avizienis, J. Laprie, B. Randell, C. Landwehr, Basic concepts and taxonomy of dependable and secure computing, IEEE Trans. Depend. Secur. Comput. 1 (1) (2004).
[22] D.K. Pradhan, Fault-Tolerant Computer Systems Design, Englewood Cliffs.
[23] W.R. Dunn, Practical Design of Safety Critical Computer Systems, Reliability Press, Solvang, CA, 2002.
[24] Y. Jiang, C. Tham, C. Ko, Challenges and approaches in providing QoS monitoring, Int. J. Netw. Manage. 10 (6) (2000) 323–334.
[25] D. Avresky, J. Arlat, J.C. Laprie, Y. Crouzet, Fault injection for formal testing of fault tolerance, IEEE Trans. Reliab. 45 (3) (1996) 443–455.
[26] Guan, Qiang, Song Fu, Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures, in: Proc. of IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS), 2013.
[27] Chuah, Edward, et al., Linking resource usage anomalies with system failures from cluster log data, in: Proc. of IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS), 2013.
[28] W. Chen, S. Toueg, M. Aguilera, On the quality of service of failure detectors, IEEE Trans. Comput. 51 (1) (2002) 561–580.
[29] R.P. Jagadeesh Chandra Bose, S.H. Srinivasan, Data Mining Approaches to Software Fault Diagnosis, in: 15th International Workshop Research Issues in Data Eng.: Stream Data Mining and Applications, 2005, pp. 45–52.
[30] L. Cherkasova, K. Ozonat, N. Mi, J. Symons, E. Smirni, Anomaly? Application change? or workload change? Towards automated detection of application performance anomaly and change, in: Proc. IEEE International Conference on Dependable Systems and Networks, 2008, pp. 452–461.
[31] IEEE Std 1044-1993, IEEE Standard Classification for Software Anomalies, 1993.
[32] A. Bondavalli, A. Ceccarelli, F. Gogaj, A. Seminatore, M. Vadursi, Experimental assessment of low-cost GPS-based localization in railway worksite-like scenarios, Measurement 46 (1) (2013) 456–466.
[33] A. Bovenzi, F. Brancati, S. Russo, A. Bondavalli, An OS-level framework for anomaly detection in complex software systems, IEEE Trans. Dep. Sec. Comput. 12 (2015) 366–372.
[34] S.M. Kay, Fundamentals of statistical signal processing, Detection Theory, vol. II, Springer, 1998.
[35] A. Ceccarelli, T. Zoppi, A. Bondavalli, F. Duchi, G. Vella, A testbed for evaluating anomaly detection monitors through fault injection, in: Proc. of 2014 IEEE 17th International Symposium On Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 10–12 June 2014, pp. 358–365.
[36] A. Ceccarelli, T. Zoppi, M. Leone Itria, A. Bondavalli, A multi-layer anomaly detector for dynamic service-based systems, in: Koornneef, Floor, van Gulijk, Coen (Eds.), Computer Safety, Reliability, and Security (SAFECOMP). Lecture Notes in Computer Science, vol. 9337, 2015, pp. 166–180.