This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

Impact of Failure Prediction on Availability

Modeling and Comparative Analysis of Predictive and Reactive Methods

Igor Kaitovic, Student Member, IEEE, and Miroslaw Malek, Senior Member, IEEE

Abstract— Predicting failures and acting proactively have a potential to improve availability as a correct prediction and a successful mitigation may bring a reward resulting in decrease of downtime and availability improvement. But, conversely, each incorrect prediction may introduce additional downtime (penalty). Therefore, depending on the quality of prediction and the system parameters, predictive fault-tolerance methods may improve or may degrade availability in comparison to the reactive ones. We first derive taxonomies of fault-tolerant techniques and policies to differentiate between reactive and proactive policies that are further classified as systematic and predictive. To evaluate whether a predictive policy improves availability or not, we derive an analytical model for availability quantification. We use Markov chains to extend steady-state availability equation to include: precision and recall, penalty and reward, mitigation success probability and potential failure rate increase due to the prediction load. We also derive A-measure to optimize failure prediction for increasing availability. In our conclusion, precision and recall have comparable impact on availability as changing MTTF and MTTR. To validate the model we also simulate and analyze availability of a virtualized server with exponential distribution of failure and repair rates.

Index Terms— Availability, failure prediction, fault tolerance, modeling, optimization, predictive, proactive

1 Introduction

FAILURES degrade system's availability and cause service interruptions. The consequences include user dissatisfaction, loss of business opportunities, and increase of operational cost or they may even be more severe in a safety-critical environment. As a remedy, methods to attain dependability have been proposed including different fault-tolerance policies. Fault tolerance (FT) improves availability by allowing continues operation in the presence of faults typically by masking a failure by redundancy (e.g. Triple Modular Redundancy), reactively recovering after a failure or by mitigating failures proactively. Since masking by massive redundancy is usually too expensive we focus on reactive (corrective) and proactive methods.

With ever-growing need for more functionality and performance, the complexity and the size of ICT systems, for example, servers and computer clouds, increase rapidly. Typically, an increase in the number and new types of faults and, consequently, a higher failure rate can be observed. For example, about 1,000 of individual machines of a Google cloud cluster with 1,800 servers fail per year. On the average, this makes almost one server failure per 8 hours [1]. On the other hand, with proliferation of ICT services to all walks of life and growing users' reliance on services and systems, keeping high level of availability of such systems is even more important. Reactive FT policies might be easier for implementation but may not suffice to ensure availability of large and complex systems [2]. In fact, it will be essential for the future systems resilience to take into account regular component failures as they are expected to increase significantly [3]. Therefore, there is a need for more advanced policies capable to surpass the current level and to further enhance availability.

Proactive fault-tolerance policies promise higher availability and overall dependability improvement. This is achieved by taking actions to avoid (prevent) or to mitigate failures before they occur. For example, proactive actions may be taken when a failure is predicted with online failure prediction methods. This approach is already being taken by leading enterprises, such as HP to ensure reliability, availability, and serviceability in exascale computing in the face of hardware failures [4]. A comprehensive survey and a taxonomy of failure prediction methods that may be used for this purpose is presented in [5]. To differentiate between the cases when proactive mitigation of failures is based on best practice and expected failure rate (e.g. periodic checkpointing) and when it is triggered by failure prediction, we refer to the latter as *predictive fault-tolerance*. Terms, proactive [2], proactive fault handling [6] and adaptive fault tolerance [7] are also used in the literature with the same meaning.

Proactive action for failure mitigation may also introduce some downtime that we refer to as *proactive action overhead*. Typically, the overhead is much less than the downtime caused by a failure and system recovery. For example, proactive rejuvenation may be performed during a system idle time and will cause less or no downtime than

- *I. Kaitovic is with the Advanced Learning and Research Institute (ALaRI), Faculty of Informatics, Università della Svizzera italiana, Via Giuseppe Buffi 13, 6900 Lugano, Switzerland. E-mail: igor.kaitovic@usi.ch.*
- *M. Malek is professor and director of the Advanced Learning and Research Institute (ALaRI), Faculty of Informatics, Università della Svizzera italiana, Via Giuseppe Buffi 13, 6900 Lugano, Switzerland. E-mail: miroslaw.malek@usi.ch.*

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

the reactive one [7]. Thus, assuming that a failure is predicted correctly and mitigated successfully, downtime with predictive policy will be shorter than the reactive policy downtime and availability will improve. We refer to this downtime decrease as a *reward*. Nevertheless, as there is no perfect oracle, not every failure can be predicted. Also, there is no guarantee of a successful mitigation. For example, if a system is already in a contaminated state when a failure is predicted, the failure might be unavoidable. Furthermore, a failure might be (incorrectly) predicted even when it is not imminent. Such a false alarm will still trigger a proactive action that will introduce unnecessary overhead that we call *penalty*. Finally, as also observed in [2], prediction introduces additional computational load that may increase frequency of failures [8].

In conclusion, on one hand, predictive fault tolerance may enhance availability but, on the other hand, high overhead, load increase, unsuccessful mitigation and low quality of failure prediction may even result in availability decrease. For example, it has been observed in [9] that a predictive policy may increase a task execution time by 10% when failure prediction quality is low.

Despite a number of implementations of predictive fault tolerance, which we review in the next section, there is still a need to further investigate a tradeoff between reactive and predictive methods and to provide an analytical model for availability with predictive FT. In particular, it is not clear to what extent availability may be enhanced with a predictive FT, how this effect can be modeled, what the minimum requirements for the prediction quality are (in terms of precision and recall), and how failure prediction can be optimized so that availability enhancement is maximized. We address this problem in our previous work [6], [10] but only in part and we extend the approach here.

Our main goal is to provide a comprehensive and practical, analytical model of predictive FT that may be used to quantify availability. This helps designers to quickly estimate availability with predictive and reactive policy and to select the optimal one for the improving availability of a specific system with available failure prediction mechanisms. We start by analyzing the existing fault-tolerance techniques and policies to extend the fault-tolerance taxonomy proposed in [11]. We identify two types of predictive fault tolerance and create a specific Markov chain model for each one. To model availability in a comprehensive way we create a unified Markov chain that incorporates both types of predictive policies and that includes the following parameters: *(i) precision* and *(ii) recall* of failure prediction, *(iii) penalty* and *(iv) reward*, *(v) proactive action success probability* and *(vi) failure rate increase* due to additional load introduced by prediction. Using the model we analyze the effect of a predictive FT on availability and extend the steady-state availability equation so that it includes the model parameters. As, in some cases, it may be possible to change the quality of failure prediction by manipulating prediction parameters, we derive an A-measure to optimize failure prediction for maximizing availability. We also provide a guideline for availability equation and the A-measure application. We present a case study in the scope of which we analyze availability improvement of a virtualized server with predictive FT and we conduct sensitivity analysis of the equation and the Markov model to identify the variables that affect availability the most. Finally, to validate the model and to demonstrate how reactive or predictive policy may be more appropriate depending on the system parameters, we simulate a virtualized server that supports live migration, when failures follow exponential and Weibull failure distribution and when repair rates follow exponential and lognormal distributions. We quantify availability for the two policies as the proactive action overhead increases.

The rest of the paper is organized as follows. In the next section we discuss related work. In Section 3 we present taxonomies of fault-tolerance techniques and policies. Predictive fault-tolerance policies are described in detail in Section 4. Steady-state availability equation and A-measure are derived in Section 5, and a guideline for their application is described in Section 6. A case study is presented in Section 7 and Section 8 concludes the paper.

2 Related Work

The use of predictive fault-tolerance policies is gaining popularity, especially in cloud and HPC systems communities [4]. We review some of these policies and frameworks for their modeling and evaluation to identify current trends, application fields, and properties.

A quantitative evaluation of the impact of proactive software rejuvenation on downtime and availability of cluster systems has been presented by Castelli *et al.* in [12]. The authors have considered two types of rejuvenations: time-based (periodic) and prediction-based (predictive). Simulation results indicate that, depending on the system configuration, periodic rejuvenation may reduce the total downtime by more than 25% when compared to the same system without rejuvenation, whereas predictive policy with 0.9 recall (that is referred to as prediction coverage) may achieve 60% of downtime reduction. The authors assume low false-alarm rate but acknowledge that too frequent rejuvenations may introduce significant cumulative downtime even when the rejuvenation overhead is low.

Assuming that periodic virtual machine (VM) migration rejuvenates Virtual Machine Monitor (VMM) and improves availability, Melo *et al.* propose an approach that combines software rejuvenation with VM migration and compare two different types of migration scheduling in [13]. Using reliability models on a test system, they conclude that availability may be enhanced when an optimal migration period, that is very sensitive on the migration overhead, is combined with coordinated checkpointing.

A proactive VM migration-based approach for decreasing wall-clock execution time and financial cost of renting computational nodes in a cloud environment has been presented by Egwutuoha *et al.* in [14]. It is based on detecting unhealthy nodes using run-time values of parameters like CPU temperature and fan speed. Detection of an unhealthy node triggers acquisition of a new node and migration of VMs. Experimental results indicate the maximum of 30% decrease in an application execution time. A similar approach has been taken in [15] by Nagarajan *et al.*, where a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

proactive migration of VMs is performed, based on the current health status of nodes and load distribution. Live and stop-and-copy migrations are used, and observed migration overheads are between 1 and 16 seconds.

In [9], Lan and Li describe FT-Pro, an adaptive approach that combines proactive migration with reactive checkpointing by making runtime decisions in response to the probability of a near-future failure. Their case study, based on a stochastic model and simulation indicate that, when compared to periodic checkpointing, FT-Pro may decrease the application wall-clock execution time by up to 43%. The gain depends on the quality of prediction and system configuration. For the model selected for the sensitivity analysis, FT-pro decreases application execution time by almost 27% when prediction is perfect (both precision and recall are 1) but increases it by almost 10% when prediction quality is low (both precision and recall are set to 0.1). This depicts the effect of prediction quality on the policy efficiency very well. Sensitivity analysis indicates that the policy is more robust to the variations of precision than to the variations of recall.

Bouguerra *el al.* [16] combine periodic (that is referred to as proactive in the paper) and prediction-based preventive (predictive) checkpointing to boost the efficiency of a high performance computing system by 30% with only up to 6% overhead when compared to purely periodic checkpointing. The used prediction method has high precision of 0.9 and recall of 0.5. The authors find that, for the specific configuration, the impact of precision is negligible due to low overhead of the implemented checkpointing.

Another adaptive approach that combines proactive and reactive checkpointing based on failure prediction (that is referred to as fault prediction in the paper) has been proposed by Aupy *et al.* in [17]. The effect of prediction quality is also addressed. Simulation results indicate that failure prediction is not always helpful but criterion on deciding between reactive and proactive strategy is not given in an analytical form. Moreover, the proposed model, that is also very complex, focuses on the specific checkpointing strategy and thus does not generalize for other strategies that are not based on checkpointing.

In [2], Eckart *et al.* use Markov models to demonstrate the effect of failure prediction, combined with proper reactive measures, on mean-time-to-data-loss (MTTDL) of storage systems. Modeling different RAID configurations, they find that MTTDL can be extended by 85% when prediction recall equals 0.5 assuming that other idealistic conditions hold (e.g. negligible number of false alarms).

Vallée *et al.* proposed a generic framework [18] and a simulator [19], for modeling and simulation of various fault-tolerance policies and their evaluation with respect to execution overhead. The impact of a percentage of predicted failures (which is equivalent to recall) and checkpoint interval on execution overhead may be measured in the framework. Unfortunately, it does not consider the effect of incorrect predictions and does not incorporate availability evaluation methods. Another architectural blueprint for managing dependability of server systems in a proactive fashion and maintaining high availability has been proposed by Polze *et al.* in [20]. The paper emphasizes the effect of the number of dirty pages on the migration overhead that further decreases availability.

In [6], Salfner and Malek propose an extension of availability equation to demonstrate how availability may be enhanced with predictive approach (that is referred to as proactive in the paper). The extended equation, includes mean-time-to-failure (MTTF) and mean-time-to-repair (MTTR) and five additional measures: precision, recall, prevention probability, repair time improvement and the risk of introducing additional failures. It is assumed that availability may be affected only when the prediction is correct so the effect of false alarms on availability is not considered. Still, the authors point out that prediction may have a negative effect on availability but only by introducing additional failures due to load increase and do not consider false-alarms. The authors do not provide guidelines and measures for optimizing failure prediction to maximize availability.

Reviewed research papers clearly demonstrate the benefits of predictive approaches versus reactive ones. The negative effect of false alarms is identified in some of the papers. Still, a comprehensive model and methods for optimizing failure prediction and evaluating the effect of predictive FT on availability have not been proposed.

3 Taxonomy of Proactive Fault-tolerance Techniques and Policies

In [11], Avižienis *et al.* described basic dependability concepts and also defined a taxonomy of fault-tolerance techniques. We extend this taxonomy to include techniques of proactive fault tolerance by considering those presented in the papers reviewed in Section 2. The aim is to support differentiation between FT polices and to help to understand important properties of the predictive ones. The extended taxonomy is presented in Fig. 1. Parts that are present in the original taxonomy from [11] are written in italics, and classes of techniques are presented in rectangles.

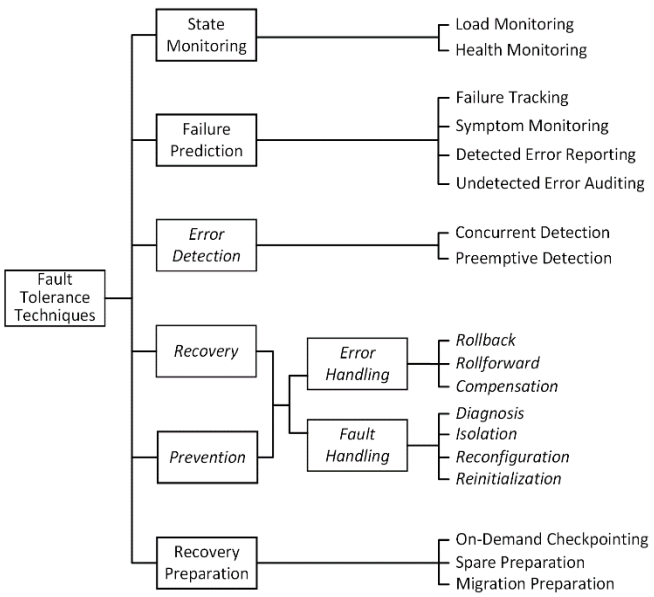


Fig. 1. Taxonomy of fault-tolerance techniques.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

In system state monitoring techniques either the health-status (as, for example, in [14]) or the load (as, for example, [21]) of the system are estimated by monitoring system parameters (features or variables) such as fan speed, memory usage, CPU activity and temperature, disk activity, and number of exceptions. A similar set of parameters can be monitored to anticipate upcoming errors (e.g. component failures). In [5] four main types of failure prediction techniques are identified: failure tracking, symptom monitoring, detected error reporting, and undetected error auditing. Error detection may be performed concurrently with the execution of the main task or preemptively when the execution of the main task is suspended (e.g. memory checking or spare checking) [11]. A good survey of error detection mechanisms is presented in [22]. The same set of error- and fault-handling techniques that are identified in [11] as system recovery techniques, may be used to prevent errors and failures when triggered based on the results of the state estimation or failure prediction. When a failure is anticipated, the system can be prepared for the recovery by creating a checkpoint on-demand (as in [9]), acquiring a spare node (as in [14]), preparing a VM for migration or using similar techniques.

Considering only techniques originally presented [11], the authors of the same paper have identified two types of reactive fault-tolerance policies: *detection and recovery* and *masking and recovery*. In the first one, recovery actions are triggered on demand, when a failure of a component is detected. In the second policy, errors (component failures) are systematically masked (for example with triple modular redundancy) and recovery is performed on demand when an error is detected. Note that different implementations of these basic policies may also include additional, proactive actions that are performed systematically. For example, periodic checkpoints are created as a part of the rollback recovery policy that is one possible implementation of a detection and recovery policy type.

Taking into account all FT techniques from Fig. 1, we identify additional types of FT policies that we present in a form of taxonomy in Fig. 3.

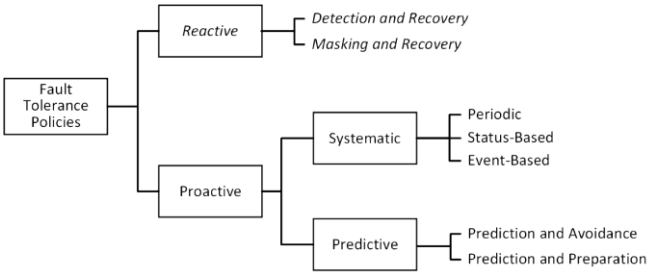


Fig. 3. Taxonomy of fault-tolerance policies.

Fault-tolerance policies can be *reactive* or *proactive*, with reactive being the ones already defined in [11]. Proactive policies can further be categorized as *systematic* or *predictive*. Systematic policies include those where actions are taken periodically, with a period defined statically or adjusted dynamically as in [21], depending on the system status indicators as, for example, system health and load distribution (e.g. as in [15]), or after specific events (e.g. a completion of a task or a preemptive error detection).

Predictive policies are based on prediction of near-future failures. If a failure is predicted, either an action to avoid the failure or to prepare the system for recovery may be triggered. Depending on this, we identify two types of predictive policies: (a) *prediction and avoidance,* and (b) *prediction and preparation.* To explain them better, we depict system availability states for the two policies in Fig. 2. The system is available when providing the intended service. It is unavailable when under recovery or during a period that corresponds to the proactive action overhead. Lightning symbol in the figure indicates a failure occurrence time and an alarm (bell symbol) indicates a failure prediction time. For simplicity, we assume that a proactive action is initiated as soon as a failure is predicted. For comparison, states for the case of the reactive policy are also depicted.

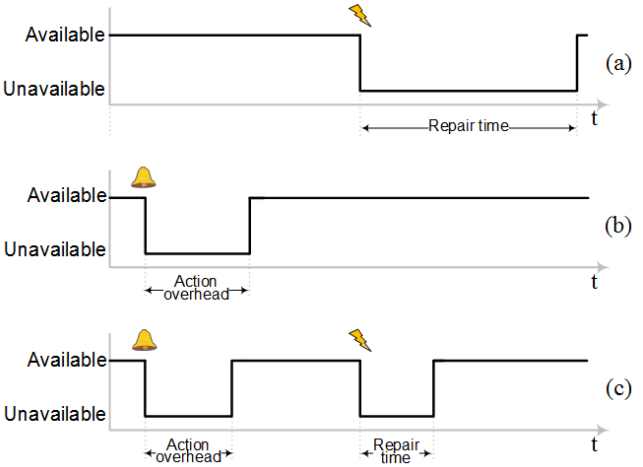


Fig. 2. System availability states for the case of a correct prediction and successful proactive action for (a) reactive, (b) prediction and avoidance, and (c) prediction and preparation fault-tolerance policies.

In the case of the reactive policy, a failure causes a downtime that is equal to the system's repair time (including failure detection, fault localization and correction).

In the case of *prediction and avoidance* policy, a proactive action to avoid the failure is initiated when a failure is predicted. As it introduces an overhead (downtime), the action should be such that the overhead is shorter than the repair time. That way, the total downtime associated with the failure will be shorter than in the case of the reactive policy and availability will improve. For example, if a failure of a node is predicted, it may be avoided by migrating a VM to another node (e.g. as in [14]). Migration introduces interruption of the main process (as explained in [20]) but the associated downtime is still significantly shorter than the time needed to repair the node or to migrate and restart the process on another node after the failure.

When *prediction and preparation* policy is used, a proactive (preparatory) action may come with an overhead but the time needed to repair the system after the failure will be shortened so that the total downtime is less than in the case of reactive policy (thus availability improves). For example, a cold spare node may be warmed up when a failure is predicted so that the time for restarting a VM after the migration is less than in the case of a reactive policy.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

Another example is on-demand checkpointing (e.g. as in [17]). Making a checkpoint when a failure is predicted shortens the repair time if the failure really occurs, as the checkpoint is closer in time to the failure occurrence and thus there is less time spent to recompute after the recovery.

In the previous paragraphs we have implicitly introduced a few simplifications that make predictive policies almost perfect and thus always superior when compared to the reactive or systematic ones. We have assumed that prediction is performed well in advance so that there is sufficient time to perform a proactive action. This is a reasonable assumption as in another case the predictive policy makes no sense. However, we will address this assumption in a more formal way in the next section. More importantly, we have assumed that failure prediction is perfect in the sense that all failures are predicted and that there are no false alarms. Obviously, this is not the case in practice but it also does not imply that a predictive policy with an imperfect prediction cannot be used to improve systems' availability. This raises questions, such as: How does prediction quality affect system's availability with a predictive policy? For what prediction quality is availability improved with respect to the reactive policy? Can we optimize prediction for maximizing availability? These are the central questions that we give answers to.

4 Predictive Fault-Tolerance Policies Model

To derive a model of predictive fault-tolerance policy we first create models of failure prediction and proactive actions. We use these models to generate Markov models for the two types of predictive FT policies. We then derive a generic Markov model of a predictive FT that we use to extend steady-state availability equation. We also briefly describe model properties that are not directly impacting the availability but are still important to fully understand the concept of failure prediction and predictive FT. This includes constraints and conditions that must be met to make predictive FT a valid policy.

4.1 Failure Prediction Model and Quality Metrics Equations

The goal of a failure prediction is to predict, with sufficient lead time, whether a failure will occur in a certain time period that is referred to as the prediction window. A predictor should predict as many failures as possible while keeping the number of incorrect predictions to the minimum. The output of a failure prediction can be categorical (Boolean) if it forecasts whether a failure will occur or not, or numerical if a probability of a failure imminence is estimated. By setting a prediction threshold, so that a failure is predicted when the probability is above the threshold, a numerical output can be translated into categorical.

Depending on the result of failure prediction and its actual occurrence in the prediction window, a prediction may be true-positive (when a failure is predicted and it also occurs), false-positive (when a failure is predicted but none occurs), true-negative (when no failure is predicted and none occurs) or false-negative (when no failure is predicted but it occurs). This is also summarized in Table I.

TABLE I
Failure Prediction Contingency Table

		Observation	
		Failure	No failure
Prediction	Failure	True positive	False positive
	No failure	False negative	True negative

Different metrics may be used to evaluate failure prediction but, as failures are still relatively rare events, precision and recall are identified as the most appropriate one. If the total number of true-positive, false-positive, true-negative, and false-negative predictions is contributed by n_{tp}, n_{fp}, n_{tn}, and n_{fn} respectively, the total number of failures is n_f, and the total number of alarms n_a, then precision (P) and recall (R) may be defined as in (1) and (2). If n_{tp} equals zero, both precision and recall are zero. If the number of alarms is zero, precision is set to one by convention. If the number of failures is zero, recall is undefined. The last case will not be considered as a hypothetical system that never fails is fault-free and needs no fault tolerance.

$$Precision = \frac{n_{tp}}{n_a} = \frac{n_{tp}}{n_{tp} + n_{fp}} \quad (1)$$

$$Recall = \frac{n_{tp}}{n_f} = \frac{n_{tp}}{n_{tp} + n_{fn}} \quad (2)$$

In an ideal case, both precision and recall are equal to 1. In practice, they are related and one measure may frequently be improved at the expense of the other. In fact, for different predictor configurations, different precision-recall pairs can be obtained but, as also observed in [23], prediction with high precision usually has low recall, and vice versa. The relation may be depicted in a precision-recall curve as the one in Fig. 4.

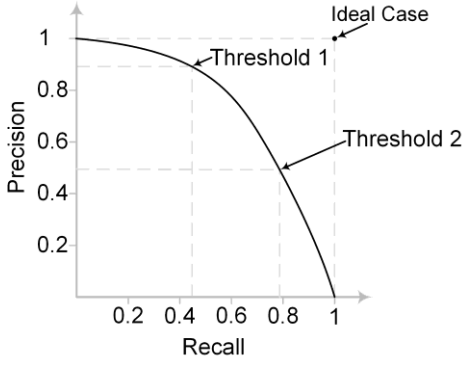


Fig. 4. An example of a precision-recall curve.

When the output of a prediction is numerical, the relation between precision and recall can be tuned by changing the prediction threshold. Typically, higher threshold causes higher precision and lower threshold causes higher recall. In Fig. 4, Threshold 1 is greater than Threshold 2.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

A relative importance of one measure over the other depends on the application. For example, in a safety-critical environment, predicting as many failures as possible is more important than having fewer false-alarms as each failure may lead to a fatal consequence. In this case, increasing recall has priority over increasing precision. When actions that are performed upon failure prediction are expensive (in a sense of introduced overhead or actual cost), it is more important to have high precision because it means fewer false alarms.

Finding an optimal precision-recall trade-off is most frequently done with F-measure, which is a harmonic mean of precision and recall. As we will demonstrate in our case study, F-measure might not be the most appropriate when the objective is availability maximization as it ignores the field of prediction application, and assumes equal importance of precision and recall.

It is also important to observe that, if a predictor is running on the system for which it is predicting failures, then it may also introduce additional load that depends on the computational complexity of the prediction algorithm. With the load increase, failure rate increases as well [24]. To capture failure rate increase caused by higher load we introduce failure rate increase factor as a parameter of failure prediction.

Thus, we model failure prediction with precision, recall and failure rate increase to higher computational load.

4.2 Proactive Action Model

Proactive actions are taken either to avoid failures or to decrease the failure recovery time by preparing for it in advance. A proactive action may be modeled with action latency, action overhead, and success probability.

Latency is the total time needed to perform an action and the overhead is the time during which the execution of the main process is interrupted. Thus, the overhead causes downtime. Latency does not affect availability directly but is important parameter as action latency should always be less than prediction lead time. Otherwise, predictive policy makes no sense as the failure will occur before a proactive action has been completed. For example, in [20] two stages of live VM migration are identified: preparatory and blackout. In the preparatory stage, parts of VM are migrated while keeping the VM running on the original node, and in the blackout stage, dirty pages are copied from the source to the target node while interrupting VM execution. In this case, latency is the sum of the preparatory and the blackout stage time, and overhead corresponds to the blackout stage.

Finally, there is no guaranty that a proactive action will be successful. For example, a system may already be in a contaminated state when a failure is predicted and the failure will occur despite the attempt of avoidance. Success probability describes a probability that a failure is avoided or that the system is prepared for the recovery with a proactive action.

4.3 Models of Predictive Fault-Tolerance Policies

Previously identified parameters of failure prediction and proactive action are used to derive models of predictive fault-tolerance policies.

A continuous-time Markov chain (CTMC) model of a prediction and avoidance fault-tolerance policy is presented in Fig. 5a. Parameters λ, μ, P, R, a, and c stand for: failure rate, repair rate, precision, recall, failure rate increase factor, and proactive action success probability, respectively. Failure and repair rates are those of the system with only reactive policy and equal 1/MTTF and 1/MTTR, respectively where MTTF and MTTR are mean-time-to-failure and mean-time-to-repair with a reactive policy. Parameter α is the alarm rate and can be expressed as (R/P)(1+a)λ, whereas γ is proactive action completion rate that equals 1/overhead.

Unpredicted failures bring the system to the Repair state with the rate (1-R)(1+a)λ. Alarms (true-positive and false-positive predictions) bring the system to the Avoidance state with the rate α. A transition from the Avoidance state to the Up state occurs at the rate (1-P)γ + cPγ. This corresponds to the cases of false-positive predictions and successful failure avoidances when a prediction is true-positive. When a failure is correctly predicted but unsuccessfully avoided, a transition from the Avoidance to the Repair state occurs at the rate P(1-c)γ. We neglect transitions from the Avoidance state to the Repair state for the cases when another failure occurs during the proactive action overhead time assuming γ>>λ.

A CTMC model of a prediction and preparation fault-tolerance policy is presented in Fig. 5b. The difference with respect to the model from Fig. 5a, apart from the naming of the states, is that correct prediction and successful preparation brings the system to the Prepared Repair state with rate cPγ. As the system is prepared for the recovery, mean time spent in the Prepared Repair state ($MTTR_{pr}$) is shorter than MTTR. The rate δ, that is a transition rate from the Prepared Repair state to the Up state, equals $1/MTTR_{pr}$.

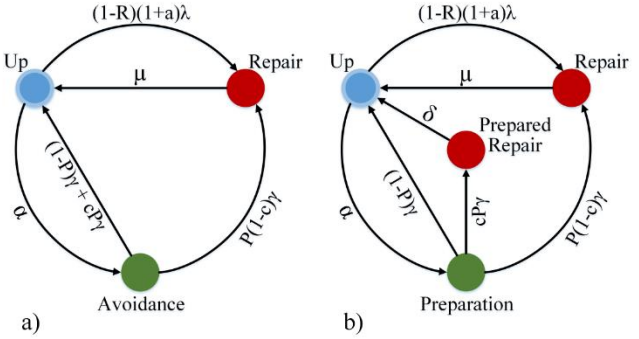


Fig. 5. Markov models of two fault-tolerance policies (a) prediction and avoidance and (b) prediction and preparation.

Solving the models from Fig. 5 for the system steady state, one can derive two different steady-state availability equations for the two types of predictive FT. Deriving a generic availability equation requires to introduce two generic parameters, *penalty* and *reward*. *Penalty* corresponds

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

to the action overhead when the action was needless or unsuccessful and *reward* is a downtime decrease in the case of a correct prediction and successful proactive action. To describe these parameters more precisely we analyze availability and unavailability time periods of a system with reactive and the two types of predictive FT policies considering the four cases of prediction outcome (see Table I).

The case of a true-positive prediction assuming a successful proactive action has already been explained in Section 3 when introducing the two proactive policies. It is also depicted in Fig. 2. We define *reward* for the case of *prediction and avoidance* policy as MTTR – overhead (where overhead refers to its mean value). For the case of *prediction and preparation* policy, we first introduce $MTTR_{pr}$ as mean-time-to-repair when the system is prepared for the recovery and then define *reward* as MTTR – (overhead + $MTTR_{pr}$). In the case of a false-positive alarm a preventive action is still initiated even though it is needless. Performing a needless action introduces downtime (*penalty*) that is equal to the overhead. If a failure is not predicted, but still occurs (false-negative prediction), the same scenario as in the case of reactive policy applies. When prediction is true-positive but a proactive action is unsuccessful, the total downtime equals the sum of the *penalty* (that equals to the overhead) and MTTR.

With penalty and reward defined, we derive a unified Markov model of proactive fault-tolerance policies that we also compare to the models from Fig. 5 to ensure equivalence. Parameters γ and δ from Fig. 5 may be expressed in function of *penalty*, *reward* and MTTR. As before, we assume transition rates from any other state to the Up state are much higher than 1/MTTF so that additional failures do not occur while the system is unavailable. The model is depicted in Fig. 6 with states described in Table II.

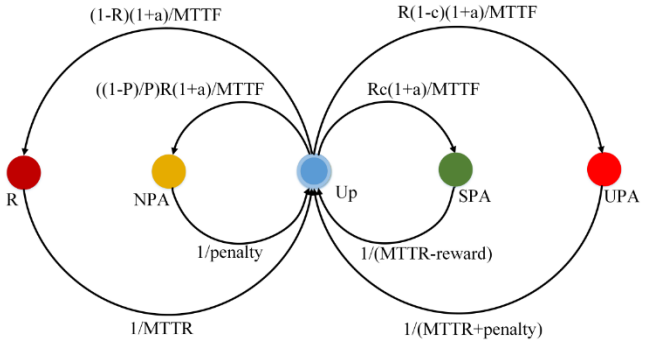


Fig. 6. Unified Markov model of predictive fault-tolerance policies.

TABLE II
DESCRIPTION OF THE GENERIC MARKOV MODEL STATES

State ID	Description
R	Recovery state.
NPA	State caused by needless proactive actions due to a false-positive alarm.
SPA	State caused by a true-positive alarm and a successful proactive action.
UPA	State caused by a true-positive alarm and an unsuccessful proactive action.

Thus, predictive FT policies may be generically modeled with *1) precision (P)* and *2) recall (R)* of failure prediction, *3) penalty (p)* and *4) reward (r)*, *5) proactive action success probability (c)* and *6) failure rate increase (a)*. In addition, the availability model has to include MTTF and MTTR of the same system when only a reactive policy is implemented.

5 AVAILABILITY EQUATION AND A-MEASURE

By analytically solving the generic model for the steady state, a steady-state availability equation is derived in (3). Availability of the same system with a reactive policy is simply the ratio between MTTF and (MTTF+MTTR). A less comprehensive version of (3) can be derived without a Markov model by only considering cases from Table I as we did in [10]. Equation (3) can be presented in a more compact and well-known form as in (4) with $MTTF_p$ defined by (5) and $MTTR_p$ defined by (6).

$$A_P = \frac{\frac{MTTF}{1+a}}{\frac{MTTF}{1+a} + MTTR - R\left(c\,r - \left((1-c) + \frac{1-P}{P}\right)p\right)} \tag{3}$$

$$A_p = \frac{MTTF_p}{MTTF_p + MTTR_p} \tag{4}$$

$$MTTF_p = \frac{1}{(1+a)} * MTTF \tag{5}$$

$$MTTR_p = MTTR - R * \left(c * r - \left((1-c) + \frac{1-P}{P}\right) * p\right) \tag{6}$$

The equations apply when precision is non-zero. The precision is zero only when the number of true-positive predictions is zero, in which case recall is zero as well. With such a prediction no failures can be anticipated, and predictive FT that relies on it makes no sense. If recall tends to zero due to a high number of failures and a small number of true-positive predictions, availability improvement will be negligible. If precision tends to zero, the expression in parenthesis in (3) becomes negative and availability decreases.

In a simplified case, when the failure rate increase factor and the success rate can be neglected (a=0 and c=1), the break-even point can easily be derived. It defines the minimum requirement for the failure prediction quality so that availability improves with respect to the system with a reactive policy, and is defined by (7). It is interesting to notice that the break-even point depends only on precision but not on recall. However, the scale at which availability improves depends on recall as well. Inequality (7) has to hold also in a realistic case (a≥0 and c≤1).

$$P \geq \frac{penalty}{penalty + reward} \tag{7}$$

Finding a trade-off between precision and recall is frequently done with F-measure. In Subsection 4.1, we have already explained why this measure is not the most appropriate for failure prediction. For the optimization of prediction to maximize availability, we introduce A-measure that is defined by (8). The measure actually represents a relative MTTR decrease with respect to a reactive policy.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

$$A_m = R\left[r\,c - \left((1-c) + \frac{1-P}{P}\right)penalty\right] \quad (8)$$

The trade-off between precision and recall should be such that A-measure is maximized in order to maximize availability assuming the *success probability* is defined and that a prediction precision-recall curve is known.

6 Guidelines for Applying the Equations

In order to apply the equations and to find an optimal precision-recall pair, model parameters have to be estimated. As it is difficult to consider all the cases and types of systems, our aim is to give a general idea on how to collect or to estimate the parameters. Accurate estimation may be challenging and it is advisable to consider a range of parameters' values or to perform sensitivity analysis to understand to what extent estimation inaccuracy may affect the availability model. Whenever possible parameters should be taken from a real system (the target system or a similar one). An alternative approach is to perform simulations or to take values from literature.

When the system is already operational with a reactive policy and error and parameter logs are stored, they may be used to train, evaluate and tune failure prediction. Many tools that can be used for prediction design (for example, WEKA [25]) also include features for generating PR curves for different prediction configurations. System's MTTF and MTTR may be calculated from error logs in this case as well. It may also be possible to extract *penalty* values from system logs if time needed to perform specific actions is recoded. For example, when *predict and avoid* policy is based on on-demand checkpointing, and periodic checkpointing is already used in the existing system, *penalty* may be calculated as checkpointing overhead. *Reward* may be estimated in a similar way. In the previous example, *reward* would be a difference between the system's MTTR and prediction lead time. To estimate failure-rate increase, a model from [24] may be used if computational load of prediction is known.

In the case when experiments (on a similar system) or simulations may be performed, failure and system parameter logs should be created. As in the previous case, they be used to extract MTTF, MTTR and to create prediction model. *Penalty* can be estimated as average of overhead for a large number of proactive action runs. However, one should keep in mind that such estimation may have large deviation depending on the available system's configurations. In fact, whenever possible, it is the best to estimate parameters separately for each configuration. Estimating *reward* and the success probability, requires to run experiments on the system with failure prediction. The rate increase factor can be simply derived by comparing MTTF of the system without failure prediction and with failure prediction but without taking any proactive actions. To estimate *success probability* and *reward* more accurately, a large number of true positives is required. To accelerate their estimation, failure prediction should be set such that *recall* is high. This will increase a probability of both true-positive and false-positive predictions.

We will first consider the case of a *prediction and avoidance* fault-tolerance policy. Let the total number of alarms during the experiment, which corresponds to the sum of true-positive and false-positive predictions, be n_a. Furthermore, let the total number of observed cases when a failure is predicted and it also occurs, that corresponds to a correct prediction and unsuccessful avoidance, be n_{pf}. If n_{tp} is the number of true-positive predictions during the experiment, then $n_{pf} = (1\text{-}c)n_{tp}$. As precision and recall are predefined for this set of experiments, n_{tp} can be expressed as a product of precision and n_a and the proactive action success probability derived as $c = 1 - (n_{pf}/(P \cdot n_a))$. *Reward* is simply the difference between the MTTR without failure prediction and the *penalty*. For the case of *prediction and preparation* policy, success probability can be derived in a similar manner with a difference that the case of a correct prediction and unsuccessful proactive action is recognized when the downtime associated with system repair equals MTTR. The cases when a failure is predicted and the repair time is shortened correspond to a correct prediction and successful preparation. If an average repair time for these cases is $MTTR_p$, then *reward* is MTTR-(*penalty*+$MTTR_{pr}$).

Finally, when data logs are not available and when it is not possible to perform experiments and simulations, parameters may be estimated by considering similar systems described in the literature. Different phases of recovery and proactive actions should be analyzed to calculate MTTF, MTTR, *penalty* and *reward*. We take this approach in the conducted case study.

Once the parameters are estimated, the procedure for finding an optimal trade-off, between precision and recall with respect to A-measure, is rather straightforward and similar to the one used to finding the maximum F-measure. In this case, for each point in the precision-recall curve, A-measure has to be calculated to find the maximum. When precision-recall curve can be approximated with a mathematical function, an optimal point may be found by expressing precision as a function of recall (or vice versa), inserting this relation into (8), and finding the maximum of the A-measure using a derivative.

The availability equation can be applied to the entire system or only to one or more components depending on the type of failures that can be predicted. In the first case, (3) may be applied directly. In the second case, availability model of the system has to be first derived (e.g. a Markov chain or a Reliability Block Diagram) and equivalent MTTF and MTTR of a component(s) calculated following *(5)* and *(6)*. A similar procedure applies when only a type system or component failures can be treated proactively. For example, if only software failures of a server can be predicted, then a Markov model of the system with a reactive policy can be generated so that server software failures are modeled with a separate system state with appropriate transitions rates from the "Up" state to the "Down" state that correspond to $1/MTTF_{ssf}$ and $1/MTTR_{ssf}$ ("ssf" stands for "server software failure"). Then, to evaluate availability of the system with a predictive policy, $MTTF_{ssf}$ and $MTTR_{ssf}$ have to be substituted with appropriate measures following *(5)* and *(6)* and the entire Markov model has to be reevaluated.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

7 Case Study

To demonstrate the application of the derived generic Markov model and the equations, and to analyze sensitivity of system's availability with respect to different model parameters for the two types of predictive fault-tolerance policies, we consider a simple virtualized server system.

7.1 System Structure and Parameters

As host (hardware) failures are identified to be among the most frequent and severe ones [1][26] with host's MTTF and MTTR having a large effect on availability [27], we analyze availability of the server infrastructure to understand to what extent it may be improved with predictive FT and under what conditions. The system is composed of the main server, a cold spare and a shared external storage. It also supports live virtual machine migration. The structure of the system is depicted in Fig. 7.

Initially, the system uses a reactive failover policy with checkpoint-recovery mechanism. A checkpoint is created periodically on the external storage that may be accessed from both hosts. A checkpoint saves an entire VM state. Another VM is started on to the spare host when a failure of the main host is detected, and the computation continues from the latest checkpoint. Once the failed host is repaired it takes the role of a spare. For the simplification we assume that MTTF >> MTTR and that no host failures occur while the other one is being repaired. This is also frequent practice in analysis of similar systems. In the *prediction and avoidance* policy a live migration of VM is initiated when a failure of the main host is predicted. In the *prediction and preparation* policy, a new checkpoint is created as soon as the host failure is predicted and offline migration of VM from the failed host to the spare one is conducted after a failure is detected. We assume that the lead time equals 5 minutes as in [27] that considers a commercial telecommunication server system. Also, we assume that a similar prediction method as in [27], which has low computational overhead and good prediction quality is implemented. A sufficient lead time improves proactive action success probability and low overhead ensures a small increase in failure rate. On the other hand, increased lead time may case a decrease in the quality of prediction [24].

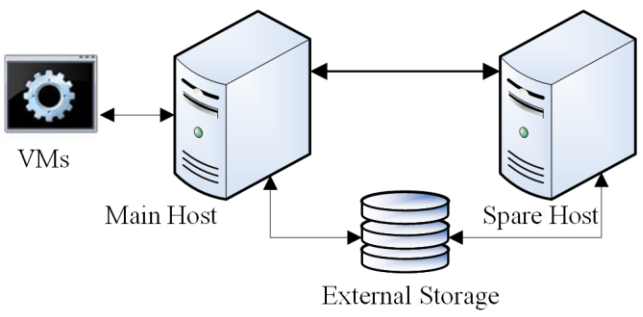


Fig. 7. Structure of the analyzed virtualized server system.

To create as realistic and as generic model of a server system as possible, host failure parameters and the success probability are adopted from [28] and adapted for the selected system, time-aspect parameters of live VM migration from [20], [29], [30] and [34], and checkpointing parameters from [31] and [32]. Other parameters estimation is based on our empirical knowledge. A generic Markov model from Fig. 6 is analyzed with the Symbolic Hierarchical Automated Reliability and Performance Evaluator (SHARPE) [33] and results are compared against (3) to ensure equivalence.

Table III summarizes system parameters. The first part of the table contains the data obtained from the literature and the calculated model parameters for the case of the reactive policy. MTTR (for the reactive policy) is the sum of mean values of failure detection time (MTTFD), migration latency (MML) and system recovery time with periodic checkpointing (MRTpcp). Predictive policies parameters are presented in the second part of the table and are calculated following the equations given in the second column. For the *prediction and avoidance* predictive policy (subscript "pa" is used to indicate policy parameters), assuming that prediction lead time is greater than the migration latency, *penalty* equals the migration overhead (that corresponds to the VM live migration blackout stage [20]), and *reward* is the difference between the MTTR and the migration overhead. For the *prediction and preparation* policy (subscript "pp" used to indicate parameters), with the same assumption for the lead time with respect to checkpoint latency, *penalty* equals checkpoint overhead (MCPO). As checkpoint is created closer to the failure, recovery time is decreased to MRT_{odcp}, and mean-time-to-repair becomes $MTTR_{pp}$. *Reward* is calculated as described in Section 4.3. It is interesting to observe that the minimum precision for which a proactive policy is superior over the reactive (the break-even point) is very low for the case of both predictive policies. The reason for this is rewards are significantly higher than the penalties.

TABLE III
System Model Parameters

	Parameter	Definition	Value
Adopted from the literature	**MTTF**	Mean-Time-To-Failure	1000 h
	MTTFD	Mean-Time-To-Failure-Detection	30 s
	MML	Mean-Migration-Latency	20 s
	MMO	Mean-Migration-Overhead	5 s
	MRT_{pcp}	Mean-Recovery-Time (periodic checkpoint)	8 min
	MRT_{odcp}	Mean-Recovery-Time (on-demand checkpoint)	4 min
	MCPO	Mean-Checkpoint-Overhead	10 s
	MTTR	= MTTFD + MML+ MRT_{pcp}	530 s
	$MTTR_{pp}$	= MTTFD + MML + MTR_{odcp}	290 s
	c	Proactive action success probability	0.90
	a	Failure rate increase factor	0.10
Derived	**$penalty_{pa}$**	= MMO	5 s
	$reward_{pa}$	= MTTR - MMO	525 s
	$P_{breakeven-pa}$	Approx. break-even point (See (7))	0.0095
	$penalty_{pp}$	= MCPO	10 s
	$reward_{pp}$	= MTTR – ($MTTR_{pp}$ + MCPO)	230 s
	$P_{breakeven-pp}$	Approx. break-even point (See (7))	0.0416

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

7.2 Availability with Reactive and Predictive Policies

Steady-state availability for the range of values of precision and recall between 0.001 and 1 is depicted for the *prediction and avoidance*, and for the *prediction and preparation* policy in Fig. 8 and Fig. 9 respectively. For the comparison, availability of the system with a reactive policy is also presented. A breakeven precision value (see (7)) is where the two surfaces in figures intersect.

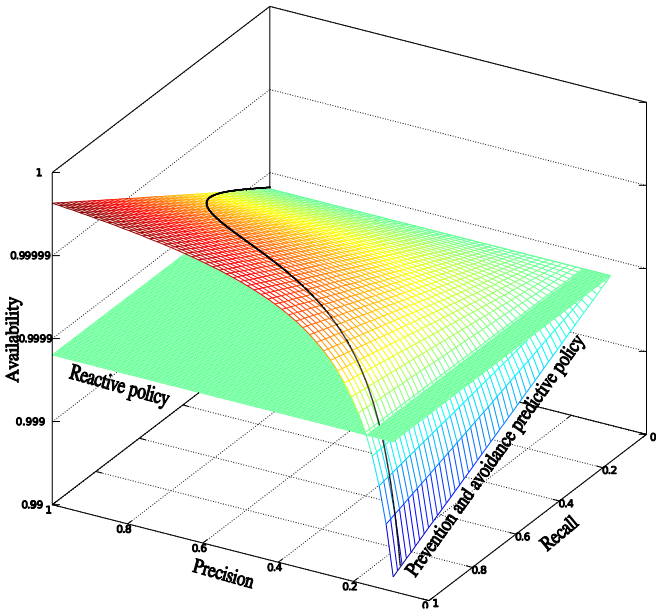


Fig. 8. Availability of the server infrastructure with prediction and avoidance, and reactive policies as a function of prediction quality. A solid line on the top surface applies to the precision-recall curve from Fig.3.

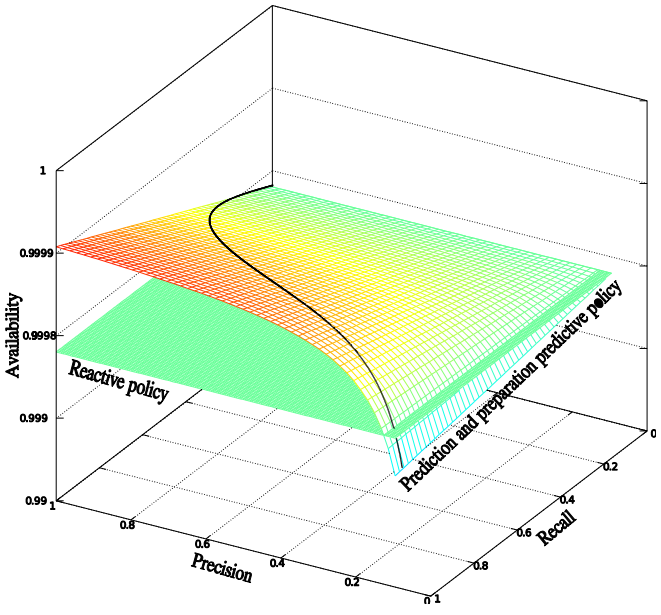


Fig. 9. Availability of the server infrastructure with prediction and preparation, and reactive policies as a function of prediction quality. A solid line on the top surface applies to the precision-recall curve from Fig.3.

Thus, when the prediction quality is such that precision is above the breakeven point – proactive policy is superior over the reactive one, and vice versa. Note that two predictive policies have different *penalty* and *reward* values, so that the range of availability that can be achieved varies differently with changing precision and recall. It is interesting to observe that the availability of a proactive system increases/decreases with the increase of recall, depending on whether precision is above/below break-even line or not.

For the selected parameters and a perfect failure prediction (P = R = 1), we compare steady-state availability, steady-state unavailability, and downtime per year for the server infrastructure with reactive and the two type of predictive policies. Result are presented in Table 4.

TABLE IV
AVAILABILITY MEASURE WITH DIFFERENT POLICIES WITH PERFECT FAILURE PREDICTION

Policy type	Availability	Unavailability	Downtime per year
Reactive	0.999853	$147*10^{-6}$	1h 17 min
Prediction and avoidance	0.999998	$2*10^{-6}$	1 min
Prediction and preparation	0.999915	$85*10^{-6}$	44 min

With the *prediction and avoidance* policy, unavailability reduces by a factor of almost 74 times, and with the *prediction and preparation* policy, unavailability is almost halved. Thus, even with highly reliable components, such as a server with MTTF of 1000 h, availability of the server system may still be additionally improved with a predictive FT policy if prediction quality is high.

The improvement effect is even more evident when reliability of the server is lower. In practice, a typical MTTF of a server is about 20 days that is, approximately, 500 h. In this case, availability with a reactive policy is 0.999705. With predictive policies, availability reaches 0.999991 for the *prediction and avoidance* policy and 0.999830 for the *prediction and preparation* policy. This corresponds to reducing downtime over one year from 2 h and 36 min, to only 4 min with *prediction and avoidance* and to 1 h and 29 min with *prediction and preparation* policy.

As a more realistic scenario, when failure prediction is not perfect, we consider the case of a precision-recall curve from Fig. 4. As an approximation, for the purpose of a simplified application of the A-measure, we assume that the curve can be represented with a function $P = 1 - R^3$.

We depict, with solid lines on the higher surfaces in Fig. 8 and Fig. 9, how availability changes with respect to the model parameters from Table III and the approximated curve from Fig. 4. In fact, the lines are projections of the approximated precision-recall curve on the availability surface. As we have already approximated the curve, optimal precision-recall trade-off may be derived by a substitution of precision in (8) by $P = 1 - R^3$, and finding a derivative of (8) with respect to recall with other parameters taken from Table III.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

Availability, unavailability, downtime per year, and downtime decrease with respect to the reactive policy, for optimal precision-recall point obtained by using A-measure are presented in Table V.

Unavailability and downtime decreases are not as high as in the case of a perfect failure prediction but significant. For example, with the *prediction and avoidance* policy unavailability reduces by a factor of almost 4.5 times, and downtime over one year is decreased by 1h when compared to the reactive policy. With *prediction and preparation* policy unavailability is decreased by almost 25% and annual downtime decrease is 17 minutes. As a comparison, we also present the values of availability measures obtained with F-measure as well as identified precision and recall values. The results clearly demonstrate superiority of the proposed A-measure.

TABLE V
AVAILABILITY FOR DIFFERENT OPTIMIZATION METRICS

		Optimization Metric	
		A-measure	**F-measure**
Prediction and avoidance	**Availability**	0.999967	0.999936
	Unavailability	$33*10^{-6}$	$64*10^{-6}$
	Downtime per year [h]	17 min	34 min
	Downtime decrease [h]	1h	43 min
	Precision	0.1672	0.6856
	Recall	0.9408	0.6800
Prediction and preparation	**Availability**	0.999888	0.999879
	Unavailability	$112*10^{-6}$	$121*10^{-6}$
	Downtime per year [h]	58 min	63 min
	Downtime decrease [h]	17 min	14 min
	Precision	0.3292	0.6856
	Recall	0.8754	0.6800

To understand how the effect of predictive policies on availability changes with server's reliability, we consider a range of MTTF from about 10 days to 100 days (more precisely from 250 h to 2500 h). Steady-state availability and downtime decrease per year are depicted in Fig. 10 and Fig. 11 for the three types of policies. Precision and recall is optimized with A-measure.

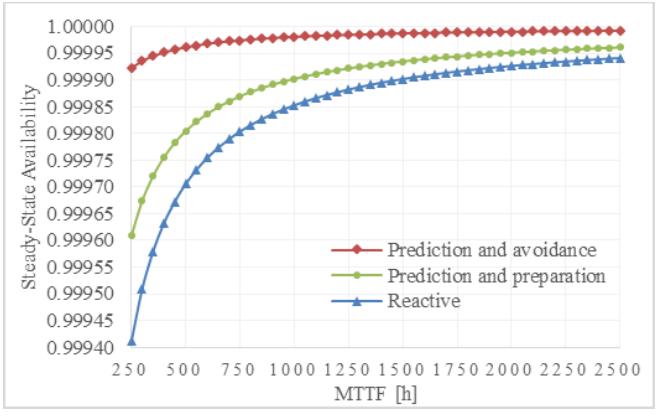


Fig. 10. Impact of MTTF on steady-state availability.

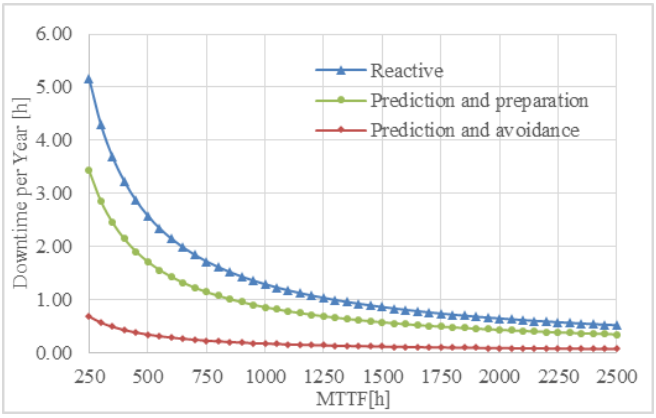


Fig. 11. Impact of MTTF on downtime per year.

7.3 Sensitivity Analysis

We further analyze the effect that changing different system parameters has on availability by conducting sensitivity analysis. We use scaled differential scaled sensitivity analysis as described in [28]. In the essence, considering a model of N input parameters x_i (i = 1 .. N) and an output Y, first a sensitivity of the output with respect to each parameter x_i, namely $S_{xi}(Y)$, is calculated as a partial derivative of Y with respect to x_i. Then, scaled sensitivity is derived by scaling sensitivity with the ratio of the nominal value of the parameter to the nominal value of the output. Scaled sensitivity rank (SS_x) for each parameter is defined as the value of a scaled sensitivity with nominal values of parameters.

Sensitivity analysis is performed for the steady-state availability of the generic Markov model of a predictive fault-tolerance policy, using a SHARPE package for the sensitivity analysis developed in the scope of [34]. The analysis is conducted with respect to the parameters: MTTF, MTTR, precision, recall, *penalty*, *reward*, *a*, and *c*. Nominal values for the parameters are adopted from Table III, and precision and recall nominal values are the ones from Table IV when A-measure is used for optimization. Sensitivity analysis of system's availability with a reactive policy with respect to MTTF and MTTR is also conducted. Values of scaled sensitivity ranks are presented in Table VI. Negative value of sensitivity rank implies that the function decreases with an increase of the parameter. A zero value implies that the output is not sensitive to the parameter change.

TABLE VI
SENSITIVITY ANALYSIS RESULTS

	Scaled Sensitivity Rank of System's Steady-state Availability		
(x)	$SS_x(A_r)$	$SSx(A_{pa})$	$SSx(A_{pp})$
MTTF	$1.15*10^{-4}$	$1.94*10^{-5}$	$9.77*10^{-5}$
MTTR	$-1.15*10^{-4}$	$-1.49*10^{-4}$	$-1.49*10^{-4}$
precision	0	$8.24*10^{-6}$	$7.83*10^{-6}$
recall	0	**$1.29*10^{-4}$**	**$5.09*10^{-5}$**
penalty	0	$-6.93*10^{-6}$	$-5.38*10^{-6}$
reward	0	**$1.36*10^{-4}$**	**$5.63*10^{-5}$**
a	0	**$-1.92*10^{-7}$**	**$-9.68*10^{-7}$**
c	0	**$1.37*10^{-4}$**	**$5.87*10^{-5}$**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2806448, IEEE Transactions on Dependable and Secure Computing

As previously observed, availability is becoming less sensitive with respect to the change in MTTF with predictive policies as the rank changes from the order of 10^{-4} with a reactive, to the order of 10^{-5} with a predictive policy. In practice, this means that a reliability of a component will affect system's availability to a lesser extent. Sensitivity with respect to MTTR remains almost the same regardless of the policy. For the both predictive policies, availability is more sensitive to the change of recall than to the change of precision. This is also in line with some implementations of predictive policies (e.g. [9] and [16]) where, based on experiments and not on formal analysis, the authors were suggesting that recall should be given priority over precision. However, one has to keep in mind that precision still has to be at least as required by (7) for a predictive policy to improve availability with respect to the reactive one. It is even more important to observe that improving recall has a comparable effect on availability as changing server's MTTF. In fact, in the case of *prediction and avoidance* policy, improving recall is even more effective than improving server's MTTF.

Adding to this a previous conclusion, that a predictive policy makes a system less sensitive to the change of MTTF and considering results depicted in Fig. 10 and Fig. 11, we may say that improving system's availability may be more effective by implementing a predictive policy with high prediction quality (when *reward* is also high) than by investing into high reliability components with low failure rate. As it could be expected, one may also observe that availability is more sensitive on recall when *reward* is higher. In fact, sensitivity to *reward* is comparable to the one with respect to recall, whereas sensitivity with respect to *penalty* is comparable to the sensitivity with respect to precision. Availability is also very sensitive to the change of a proactive action *success probability* (parameter c). One of the ways to improve this parameter is to make sure that prediction lead time is always sufficient to perform a proactive action. Increasing lead time will also decrease a probability that the system is already contaminated when a failure is predicted. On the other hand, increased lead time deteriorates the quality of prediction as observed in [24]. *Failure rate increase* due to a load introduced by failure prediction and other side effects, has a relatively low effect on availability. This means that even when failure prediction introduces significant load increase, availability may still be improved.

7.4 System Simulation and Model Evaluation

To evaluate the model, we simulate availability of the virtualized server system from Fig. 7 and compare simulation results with those obtained with the model. Failure and repair rates are modeled with exponential distribution as this is also the most frequent practice in dependability analysis. In fact, the same assumption about the exponential distribution was made in [28] from where we have adopted the model of the system.

Simulations are implemented in MATLAB and performed separately for the two types of predictive fault-tolerance with different values for the action overhead. For the *prediction and avoidance* predictive policy the mitigation overhead is set from 5 s to 530 s with steps of 5 s. For the *prediction and preparation* policy, checkpointing overhead is set from 5 s to 240 s with the same step. The upper limit for the ranges is set so that reward always remains greater than zero. When the reward is negative, reactive policy, obviously, should be used. Other parameters are the same as in the previous analysis (see Table III) with failure prediction PR-curve being depicted in Fig. 4. For each overhead value, the optimal precision and recall are calculated using (8), and penalty and reward are obtained as in Section VI. To calculate confidence intervals, for each overhead value we run 1000 simulations while also changing the random number generator settings in each run. In every run, system's availability is simulated for the lifetime of 100*MTTF (more than 11 years).

Results are presented in Figures XX and YY. The horizontal line represents availability with reactive policy. Availability estimated with our model (the extended availability equation (3)) is presented with a dashed line, whereas average availability obtained from simulations is presented with a solid line. Confidence intervals of 95%, 90% and 80% are presented with shaded areas.

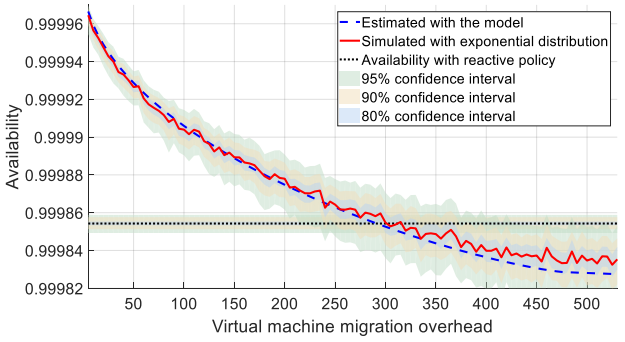


Fig. 12. System's availability with different virtual machine migration overhead for prediction and avoidance policy.

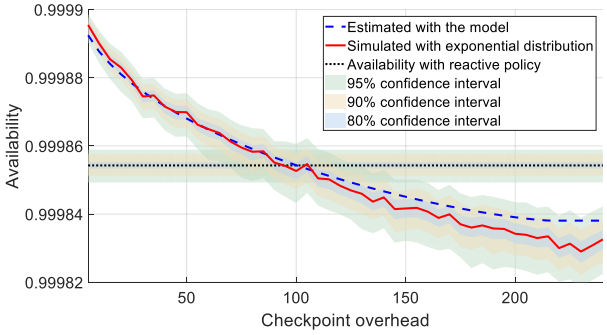


Fig. 13. System's availability with different checkpoint overhead for the prediction and preparation predictive policy.

For the *prediction and avoidance* policy, simulation results indicate that, for the specific system and selected failure predictor, it is better to use reactive policy when the overhead is over 300s. In fact, even with a perfect failure prediction it is still better to use reactive policy when the overhead is above 385s. For the *prediction and preparation* the break-even checkpoint overhead value is 115s.

The difference between availability values that are estimated with the model and the ones obtained with simulation increases as the overhead becomes higher. This is expected as, with higher overhead, the overhead variations

that also follow exponential distribution are increasing. For the same reason, simulation confidence intervals are wider when the overhead is higher. Confidence intervals for availability with reactive policy are constant as they do not depend on the overhead.

Following the simulation results, we observe that the proposed model describes availability with predictive policies well. Most importantly, selecting between the reactive and proactive policy may be done with the analytical equation (model) as minor difference between simulated and estimated availability at the break-even point, for both predictive policies, may be contributed to the simulation variance. In fact, the maximum difference in simulated and estimated availability is at the order of 10^{-5} for the whole range of the overhead.

8 Conclusions

We have analyzed the effect of failure prediction on availability and defined models for two types of predictive FT policies to answer the fundamental question whether to use reactive or predictive policy. To accomplish this we have extended the availability equation with parameters that characterize reactive and predictive FT policies. From the equation we have derived A-measure that may be used to find an optimal trade-off between the precision and recall. Using our approach, designers and system developers can decide whether to use predictive fault tolerance or not, by evaluating how effective it is for optimizing availability. We have also validated the equation by simulating availability of a virtualized server system.

The effect of prediction quality on availability is demonstrated through the analysis and is illustrated by the case study. For realistic system parameters, availability may be improved when failure prediction quality is sufficiently high. The break-even point when availability of the system is the same as with or without predictive FT, depends on precision only but the total improvement of availability is strongly affected by recall. In fact, the sensitivity of system's availability with respect to recall is comparable to the one with respect to MTTF. Therefore, it may be more cost effective to invest in high-quality prediction algorithm than in acquiring high-quality components with lower failure rates to increase availability. At the same time, we also analyze how proactive action overhead (that affects both, penalty and reward) may be decisive on whether reactive or proactive FT is better from the availability point of view.

As a part of the future work we plan to generalize the model, possibly by using phase-type distribution for the Markov model so that other distributions may be used for modeling failure and repair rates (for example, Weibull and lognormal as suggested in [26]). Moreover, we plan to include systematic proactive policies as well.

References

[1] J. Dean, "Design, lessons and advice from building large distributed systems," Keynote Address at the *3rd ACM SIGOPS Int'l Wksp. on Large Scale Distributed Systems and Middleware* (LADIS), Big Sky, MT, USA, Oct. 2009

[2] B. Eckart, X. Chen, X. He, and S.L. Scott, "Failure prediction models for proactive fault tolerance within storage systems," in *Proc. Int'l Symp. on Modeling, Analysis and Simulation of Computers and Telecommunication Systems (MASCOTS 2008)*, pp.1-8, Baltimore, MD, USA Sept. 2008

[3] D.A. Reed, and J. Dongarra, "Exascale Computing and Big Data," *Commun. ACM*, vol.58, no.7, pp. 56-68, July 2015

[4] N. Dube, and C. Strategist, "Exascale: A race to the future of HPC, The next generation of computing," Hewlett Packard Ent. Dev. LP, Palo Alto, CA, USA, Rep. 4AA6-7771ENW, Sep. 2016

[5] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comp. Surv. (CSUR)*, vol.42, no.3, art. 10, March 2010

[6] F. Salfner, and M. Malek, "Proactive fault handling for system availability enhancement," in *Proc. 9th IEEE Int'l Symp. on Parallel and Distributed Processing (IPDPS)*, Denver, CO, USA, Apr. 2005

[7] J. Alonso, J. Torres, J.L. Berral, and R. Gavalda, "Adaptive on-line software aging prediction based on machine learning," in *Proc. IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN)*, pp.507-516, Chicago, IL, USA June-July 2010

[8] R. K. Iyer, D. J. Rossetti, and M. C. Hsueh, "Measurement and modeling of computer reliability as affected by system activity," *ACM Trans. Comput. Syst.*, vol. 4, no. 3, pp. 214-237, Aug. 1986

[9] Z. Lan, and Y. Li, "Adaptive fault management of parallel applications for high-performance computing," *IEEE Trans. on Computers*, vol.57, no.12, pp.1647-1660, Dec. 2008

[10] I. Kaitovic, and M. Malek, "Optimizing failure prediction to maximize availability," in *Proc. 13th IEEE Int'l Conf. on Autonomic Computing (ICAC)*, Würzburg, Germany, July 2016

[11] A. Avižienis, J.C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," IEEE Trans. Dependable and Secure Computing, vol.1, no.1, pp.11-33, Jan. 2004

[12] V. Castelli, R.E. Harper, P. Heidelberger, S.W. Hunter, K.S. Trivedi, K. Vaidyanathan, and W.P. Zeggert, "Proactive management of software aging," *IBM J. Res. Dev.*, vol.45, no.2, pp.311-332, March 2001

[13] M. Melo, P. Maciel, J. Araujo, R. Matos, and C. Araujo, "Availability study on cloud computing environments: Live migration as a rejuvenation mechanism," in *Proc. 43rd IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN)*, pp. 1-6, Budapest, Hungary, June 2013

[14] I.P. Egwutuoha, S. Chen, D. Levy, B. Selic, and R. Calvo, "A proactive fault tolerance approach to high performance computing (HPC) in the cloud," in *Proc. 2nd IEEE Int'l Conf. on Cloud and Green Computing (CGC)*, pp. 268-273, Xiangtan, Hunan, China, Nov. 2012

[15] A.B. Nagarajan, F. Mueller, C. Engelmann, and S.L. Scott, "Proactive fault tolerance for HPC with Xen virtualization," in *Proc. 21st Int'l Conf. on Supercomputing (ICS)*, pp. 23-32, Seattle, Washington, USA, June 2007

[16] M.S. Bouguerra, A. Gainaru, L.B. Gomez, F. Cappello, S. Matsuoka, and N. Maruyama, "Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing,", in *Proc. 27th IEEE Int'l Conf. on Parallel and Distributed Processing* (IPDPS), pp.501-512, Boston, MA, USA, May 2013

[17] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing strategies with prediction windows," in *Proc. 19th IEEE Int'l Symp. on Dependable Computing (PRDC)*, Vancouver, BC, Canada, Dec. 2013

[18] G. Vallée, C. Engelmann, A. Tikotekar, T. Naughton, K. Charoenpornwattana, C. Leangsuksun, and S.L. Scott, "A framework for proactive fault tolerance," in *Proc. 3rd Int'l Conf. on Availability, Reliability and Security (ARES)*, pp. 659-664, Barcelona, Spain, March 2008

[19] A. Tikotekar, G. Vallée, T. Naughton, S. L. Scott, and C. Leangsuksun, "Evaluation of fault-tolerant policies using simulation," in *Proc. 9th IEEE Int'l Conf. on Cluster Computing (CLUSTER*), pp. 303-311 Austin, Texas, USA, Sept. 2007

[20] A. Polze, P. Troger, and F. Salfner, "Timely virtual machine migration for proactive fault tolerance," in *Proc. 14th IEEE Int'l Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, pp. 234-243, Newport Beach, CA, USA, March 2011

[21] W. Zhao, and H. Zhang, "Proactive service migration for long-running Byzantine fault-tolerant systems," *IET Software*, vol.3, no.2, pp.154-164, 2009

[22] I. Lee, and M. Sullivan, E. Krimer, D. Wan Kim, M. Basoglu, D. Hyun Yoon, L. Kaplan, and M. Erez, "Survey of error and fault detection mechanisms," *Locality, Parallelism and Hierarchy Group, Dep. of Electrical and Computer Engineering, The University of Texas at Austin*, Austin, TX, USA, Rep. TR-LPH-2011-002, 2012

[23] N. Taerat, C. Leangsuksun, C. Chandler, and N. Naksinehaboon, "Proficiency metrics for failure prediction in high performance computing," In *Proc. 25th IEEE Int'l Symp. on Parallel and Distributed Processing with Applications(ISPA)*, Taipei, Taiwan, Sept. 2010

[24] G. Horton, V.G. Kulkarni, D.M. Nicol, and K.S. Trivedi, "Fluid stochastic Petri nets: theory, applications, and solution techniques," *Eu. J. of Operational Research*, vol.105, no.1, pp.184-201, 1998

[25] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *The WEKA workbench (fourth edition)*, Morgan Kaufman, 2016

[26] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. on Dependable and Secure Computing*, vol.7, no.4, pp.337-350, Oct. 2010

[27] G. Hoffman, and M. Malek, "Call availability prediction in a telecommunication system: a data driven empirical approach," In *Proc. 25th IEEE Int'l Symp. on Reliable Distributed Systems (SRDS)*, pp. 83-95, Leeds, UK, Oct. 2006

[28] R.d.S. Matos, P.R.M. Maciel, F. Machida, D.S. Kim, and K.S. Trivedi, "Sensitivity analysis of server virtualized system availability," *IEEE Trans. on Reliability*, vol. 61, no. 4, pp. 994-1006, Dec. 2012

[29] V. Medina, and J.M. García, "A survey of migration mechanisms of virtual machines", *ACM Compt. Surv.*, vol. 46, no. 3, pp. 30:1-30:33, Jan. 2014

[30] F. Salfner, P. Tröger, and M. Richly, "Dependable estimation of downtime for virtual machine live migration," in *Int'l IARA Jour. On Advances in Systems and Measurements*, vol. 5, no. 1&2, 2012

[31] K.F. Ssu, B. Yao, and W.K. Fuchs, "An adaptive checkpointing protocol to bound recovery time with message logging," in *Proc. 18th IEEE Int'l Symp. on Reliable Distributed Systems (SRDS)*, pp. 244-252, Lausanne, Switzerland, Oct. 1999

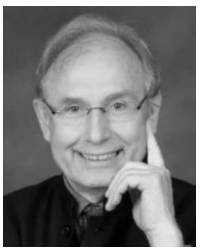
[32] N.F. Vaidya, "Impact of checkpoint latency on overhead ratio of a checkpointing scheme," *IEEE Trans. on Computers*, vol.46, no.8, pp.942-947, Aug.1997

[33] K.S. Trivedi, and R. Sahner, "SHARPE at the age of twenty two," *ACM SIGMETRICS Perform. Eval. Rev*, vol. 36, no. 4, pp. 52-57, March 2009

[34] R.d.S. Matos, "An automated approach for systems performance and dependability improvement through sensitivity analysis of Markov chains," M.S. thesis, Cent. of Inf., Fed. Univ. of Pernambuco (UFPE), PE, BR, Feb. 2011

[35] M. Mao, and M. Humphrey, "A performance study on the VM startup time in the cloud," In Proc. 5th IEEE *Int'l Conf. on Cloud Computing* (CLOUD), pp. 423-430, Honolulu, HI, USA, June 2012

Igor Kaitovic received the BS degree in computer science from the School of Electrical Engineering, University of Belgrade, Serbia in 2007, and the MS degree in embedded systems design from the Advanced Learning and Research Institute (ALaRI), Università della Svizzera italiana (USI), Lugano, Switzerland in 2010. He is currently PhD candidate at ALaRI. His expertise and main research interests include modeling, failure prediction, dependability of computer systems and smart grids as well as machine learning. He is a student member of the IEEE and the IEEE Computer Society and serves as a chair of the local student branch.



Miroslaw Malek received his MS and PhD degrees from the Wroclaw University of Technology in 1970 and 1975, respectively. From 1977 until 1994 he was Professor at the University of Texas at Austin. Then, from 1994 until 2012 he was professor and holder of Chair in Computer Architecture and Communication at the Department of Computer Science at the Humboldt University in Berlin. He is since 2012 Professor and Director of Advanced Learning and Research Institute (ALaRI) at the Faculty of Informatics at the Università della Svizzera italiana, Lugano, Switzerland. He served and serves on the number of Editorial Boards including ACM Computing Surveys. He was general chair and program chair of several ACM and IEEE conferences and workshops. He has participated in two pioneering parallel computer projects, contributed to the theory and practice of parallel network design, developed the comparison-based method for system diagnosis, codeveloped comprehensive WSI and networks testing techniques, proposed the consensus-based framework for responsive (fault-tolerant, real-time) computer systems design and has made numerous other contributions, reflected in over 250 publications and nine books. His research interests focus on dependable architectures and services in parallel, cloud, distributed and embedded computing environments including failure prediction, security and dependability.