Design-time Performability Evaluation of Runtime Adaptation Strategies (Work In Progress Paper)

Martina Rapp
Max Scheerer
FZI Research Center for Information Technology
Karlsruhe, Germany
surname@fzi.de

Ralf Reussner
Karlsruhe Institute of Technology
FZI Research Center for Information Technology
Karlsruhe, Germany
reussner@kit.edu

ABSTRACT

Performability is the classic metric for performance evaluation of static systems in case of failures. Compared to static systems, Self-Adaptive Systems (SASs) are inherently more complex due to their constantly changing nature. Thus software architects are facing more complex design decisions which are preferably evaluated at design-time. Model-Based Quality Analysis (MBQA) provides valuable support by putting software architects in a position to take well-founded design decisions about software system quality attributes over the whole development phase of a system. We claim that combining methods from MBQA and established performability concepts support software architects in this decision making process to design effective fault-tolerant adaptation strategies. Our contribution is a model-based approach to evaluate performability-oriented adaptation strategies of SAS at design-time. We demonstrate the applicability of our approach by a proof-of-concept.

CCS CONCEPTS

• **Software and its engineering → Software architectures**; **Model-driven software engineering**; **Extra-functional properties**.

KEYWORDS

performability , design-time analysis , self-adaptive systems , model-based quality analysis , Markov decision process , fault-tolerant self-adaptation strategies

ACM Reference Format:
Martina Rapp, Max Scheerer, and Ralf Reussner. 2023. Design-time Performability Evaluation of Runtime Adaptation Strategies (Work In Progress Paper). In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3578245.3585028

1 INTRODUCTION

Performability has brought together the disciplines for modeling and evaluating performance, reliability and availability attributes of software systems. Failure types (e.g. hardware, software and network failures) are known causes in performance engineering that lead to undesirable conditions or system degradation. Performability is the classic metric for performance evaluation in the presence of system failures. It allows the impact of failures on system *Quality Attributes* (QA) to be measurable and comprehensible. Well-known works in this context have focused on modeling and evaluating static software systems [9, 12].

As *Self-Adaptive Systems* (SASs) are known to operate in highly interconnected and volatile environments, we argue that they provide essential means for dealing with performability-specific quality objectives. Unlike static systems, SASs introduce challenges that one must consider during development: (*i*) the increased design space and (*ii*) the dynamic behaviour of SAS. As SASs are generalizations of static systems, the design space is expanded by SAS-specific design decisions. The dynamic behaviour of an SAS is induced by the operating environment and the adaptation strategy. SASs rely on adaptation strategies to realize runtime adaptations [7]. The changes by adaptations may have long-term effects on the quality objectives - known as *Parameters over time* [6] - that one has to consider when assessing the quality of an adaptation strategy and are hard to foresee. Performability-specific quality attributes are observed at runtime but must be already considered at design-time (DT) to make design decisions that meet the quality requirements. Thus a challenge at DT is to evaluate the effectiveness of fault-tolerant runtime adaptation strategies in situations like node failures. Poorly implemented adaptation strategies can result in operational and economic losses.

Model-Based Quality Analysis (MBQA) methods have proven successful in making informed decisions about system QA in the design phase. Various modeling and simulation tools exist that allow the prediction of performance or reliability attributes at DT (e.g. the Palladio approach [16] for static systems and SimuLizar [3] for SASs). We argue that combining established performability concepts with existing MBQA methods allow the evaluation of SAS adaptation strategies at DT to help building more resilient software systems by preventing system degradation through the design of effective adaptation strategies.

In this paper we address the **research question**: *How to determine and evaluate the effectiveness of adaptation strategies with respect to QA fulfillment in case of runtime uncertainties?* We discuss the required concepts to transfer the performability domain [9] to SASs together with the integration into an appropriate MBQA analysis framework [17] to evaluate SAS runtime adaptation strategies at system DT. The focus of [17] was in developing a framework for (i) modeling the operating environment of an SAS and (ii) an analysis approach that evaluates the quality of an adaptation strategy w.r.t. the modelled environment and specific quality objectives. This

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04…$15.00
https://doi.org/10.1145/3578245.3585028

paper reuses the concepts of [17] to evaluate adaptation strategies w.r.t. performability-specific quality objectives. We contribute to this line of work in three ways:

- C1 **Using SAS to maintain performability attributes of a system:** We generalize formally and informally existing and well-known work from the performability domain to SAS.
- C2 **Simulation of failure scenarios:** A simulation approach to predict performance attributes which reflect the quality objectives the SAS must maintain in the presence of system failures.
- C3 **Performability-specific adaptation strategy evaluation:** We integrate performability metrics from literature as performance indicators to assess the quality of an adaptation strategy.

Existing approaches use *Markov Reward Models* (MRMs) as basis for DT evaluation. However, MRMs assume a static and non-adaptive system behaviour. As for contribution C1, we expand the concept to *Markov Decision Processes* (MDPs) to account for SAS which we consider as the primary means to tackle performability requirements of a system. This serves as basis for contributions C2 and C3. Regarding contributions C2 and C3, we reuse the analysis framework of [17] by adding a novel failure scenario simulation and by integrating existing performability metrics from literature to evaluate the quality of adaptation strategies. The proposed approach allows the evaluation of SAS adaptation strategies to maintain performability-specific QAs. We prototypically implemented our approach and demonstrated its applicability with a proof-of-concept.

2 RUNNING EXAMPLE

Our running example is the *Znn.com* system which is widely used in the SAS community [5]. The system is managed by an SAS to distribute the incoming load to keep the system responsive. It consists of an adaptive load balancer and two individual application server components (see Figure 1). Each component is deployed on

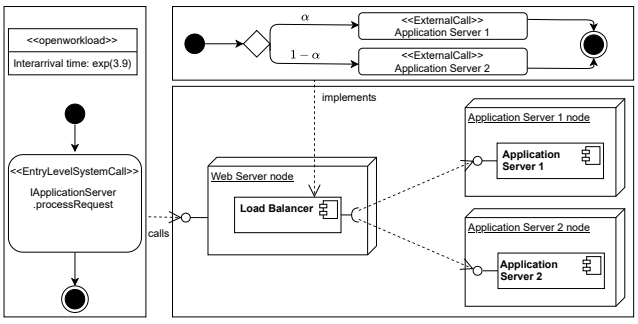

Figure 1: Znn.com-based system taken from [17]

one node. The system encounters overload scenarios with high workloads that deteriorate system performance. Since the context of this work is performability, we transfer the system into the performability domain. Both application server nodes are subject to individual node failures. Regarding performability, hardware failures are considered as a second factor, along with increased workload, that potentially degrades the performance of the system. For example, consider the situation of high workloads and the unavailability of application server 1. Thus, the adaptation problem becomes more complex because the system must not only remain responsive in high load scenarios but also deal with situations where hardware resources are not available to distribute the incoming load as best as possible.

3 PRELIMINARIES

This section introduces the foundations needed to understand the concepts described in this paper.

3.1 Mathematical framework of performability modelling

We use the mathematical framework of performability evaluation from Haverkort et al. [9] to show the mathematical consistency of our approach. The framework defines S as the set of all possible configurations in which the system can operate. The continuous-time stochastic process $X = \{X(t), t \geq 0\}$ describes the system structure (including the system components that can fail) at time t and is termed *Structural State Process*. The *Reward Rate Function* $r : S \to \mathbb{R}$ on state space S defines the steady-state performance of the system in structure state $s \in S$. r_s denotes the reward obtained from state s, i.e. $r_s = r(s)$. The reward function r is defined by multiple performance analyses, each of which is associated with a performance indicator e.g. response time (RT) or throughput. The value r_s summarizes the system performance in state s taking into account the individual performance indicators. The value of r_s is obtained by a classic performance analysis under the assumption that the system is failure-free where the system configuration reflects the components that are up and down in structure state s. The framework distinguishes four basic performability measures. Here we focus only on the steady-state performability (SSP). $Pr(s)$ is the steady-state probability of residing in state $s \in S$ at any time t. The SSP is calculated by $SSP = \sum_{s \in S} Pr(s) \cdot r_s$. The reward function r induces a *MRM* defined over process X. As X is a continuous-time Markov chain, the process $X_r = \{X_r(t), t \geq 0\}$ is denoted Markov reward process representing an MRM [10]. X_r differs from X in that it is not the structure of the considered system but the possible reward one would observe w.r.t. the system structure.

3.2 The Dynamics of Self-Adaptive Systems

We consider the dynamics of an SAS as a discrete stochastic process captured by an *MDP*. MDPs are accepted to describe the stochastic nature of SASs (e.g. [13]) and the formalization of the encountered engineering problem at SAS development [17]. An MDP is characterized by 4 elements: (S, A, t, r). S and A refer to a set of states and actions. Let s and s' be two states (i.e. $s, s' \in S$), the transition function $t : S \times A \times S \to [0, 1]$ determines the probability of transitioning from a given state s (where action $a \in A$ was taken) to state s'. The reward function $r : S \times A \times S \to \mathbb{R}$ assigns a value or reward $r \in \mathbb{R}$ to the decision of selecting action a in state s by considering state s' into which the system has transitioned. The value r is called *reward*. A policy $\pi : S \to A$ determines the action to be taken in any state. The primary objective is to find a policy that maximizes the sum of the resulting rewards.

4 APPROACH

Performability evaluation was applied for DT analyses of static systems. This does not cover structural changes of the system configuration over time resulting from actions of selected dynamic reconfigurations due to changed environmental conditions at runtime. With reference to Grassi et al. [8] we suggest to extend their work to SAS as they are a more recent representative of dynamic reconfigurable systems. We extend the traditionally used MRMs to MDPs where adaptations form the semantically equivalent concept of actions. This allows us to explore the effects of adaptation actions on system QoS attributes. Extending MRMs to MDPs requires a closer look at the following concepts: (i) formalize the extension of MRMs to MDPs in the context of SASs (ii) identify the influencing uncertainty factors in the performability domain and their representation in the SASs environment (iii) construct the reward function based on classic performability metrics (iv) integrate (i)-(iii) into a suitable MBQA analysis framework.

4.1 Extending MRMs to MDPs in the context of SASs

Originally, a structural state s (from section 3.1) defines a particular system configuration. We expand this concept by defining a state $s := (C, E) \in S$ as a tuple where C refers to the original structural state and E to an environmental state. E encompasses all variables which impact the system quality. The dynamics of an SAS can be environmental driven ($s := (C, E)$ transitions to $s' := (C, E')$), adaptation driven ($s := (C, E)$ transitions to $s' := (C', E)$ based on adaptation $a \in A$ such that $C \neq C'$) or both. Thus, the set of actions A refer to the set of adaptations of an SAS. We consider a policy π of an MDP as adaptation strategy which decides based on the current environmental state and system configuration whether an adaptation is applied or not. Opposed to Haverkort et al. [9], we associate a triple (s, a, s') with a reward r evaluated by a reward function $r(s, a, s')$ which is determined based on the performance indicators that can be derived from $s' := (C, E)$.The reward function assigns rewards to decisions made by adaptation strategy π based on performability measures.

4.2 Uncertainty factors in the performability domain

Performability literature denotes changes of the system's structure, internal state and environment as the main impact factors on the system's performance indicators [9]. Here a system consists both of the object system representing the computing system being evaluated and its operating environment (workload, external faults, etc) [12]. SASs are exposed to various sources of uncertainties classified into four groups each with a set of characteristic sources of uncertainty that relate to the: (i) system itself consisting of both the managed and the managing system, (ii) goals of a SAS, (iii) context of a SAS, (iv) humans involved in the functioning or operation of a SAS [20]. The sources of uncertainties can manifest themselves at DT or runtime. Avizienis [2] presented a common taxonomy of failure type characterization that includes hardware, software and network failures. This suggests that the influencing uncertainty factors impacting the SAS and its environment in the performability domain are the workload and the external faults that impact the system structure. A domain-specific view of the environment model with regard to failure representation consists of the relevant influential factors and conditions of failure events on the system and its QAs.

4.3 Performability-based reward function

SASs adapt themselves by selecting dedicated actions due to observed environmental changes. We need to evaluate decisions about a selected action on the system's quality attributes. To evaluate the effectiveness of adaptation strategies at DT in case of uncertain situations, we extend the originally proposed reward state function $r : S \rightarrow \mathbb{R}$ for static systems (see section 3.1) to $r : S \times A \times S \rightarrow \mathbb{R}$ for SASs (see section 3.2). Thus it is possible to evaluate not only the state but also the adaptation leading to this state. We also extend the structural system state definition of the original reward state function according to the state definition we gave. The structural system state C is expanded by the current environmental state E such that a state is defined as a tuple $s := (C, E)$. The reward assigns a value r_s to the decision of selecting an adaptation a in state s by considering the transitioned state s'. The reward value r_s is mapped to a performance indicator e.g. the system's response time (RT) in state s. As before, the reward value r_s reflects the steady-state performance of the system w.r.t. a set of performance indicators. For a given state s we run a classical performance analysis analogous as originally proposed in the performability literature. Thus, we can still use the classic performability measure of SSP to quantify the current performance level of the system after a transition by using the SSP as performability measure.

4.4 Simulation of failure scenarios

SimuLizar simulates modeled SASs [3]. During the simulation it determines QAs under load to measure performance. Errors of the simulated system cannot be taken into account by SimuLizar. Therefore we enhanced SimuLizar's analysis support to make statements about resilience of software systems at DT by injecting failures into the modeled system at specific times of the simulation. Therefore we developed a *Failure metamodel* to support the modeling of complex faults and failure scenarios with respective DT analysis by modeling failure types and failure scenarios explicitly [11]. The *Failure* metamodel follows the basic concepts and taxonomy of dependable computing as presented in [2]. It separates the *FailureType* specification from the *FailureScenario* specification (see Figure 2 and Figure 3) to describe the occurrence and effect(s) of a failure on the system. SimuLizar uses the *Failure* metamodel to generate failure events at simulation time. The changed system behavior after the occurrence of an error is also taken into account by the simulation.

FailureType: A (service) failure "is defined as an event that occurs when the delivered service deviates from correct service. The different ways in which the deviation is manifested are a system's service failure modes." ([2]). Avizienis et al. describe four different viewpoints of service failure modes that characterize incorrect services, namely (i) failure domain, (ii) detectability , (iii) consistency and (iv) consequences of failures on the environment. Thus the *FailureType* metamodel describes a failure from multiple viewpoints. It supports the explicit modeling of failures to express where, how

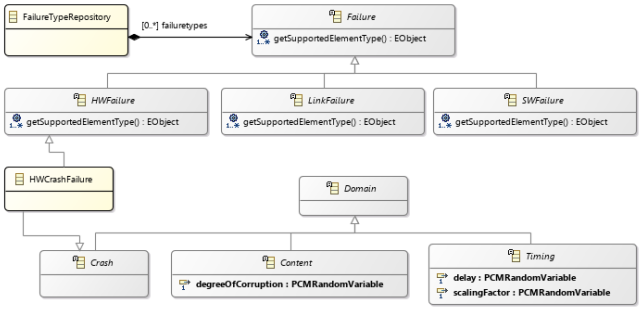

Figure 2: UML class diagram - Failure viewpoints[1]

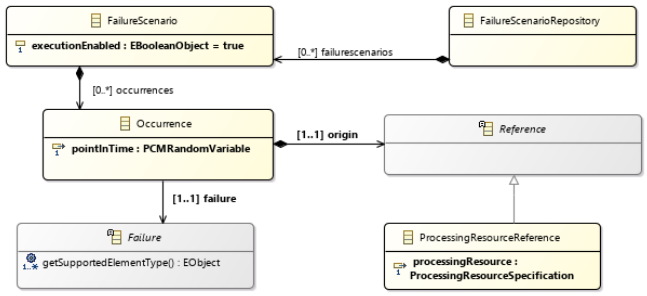

Figure 3: UML class diagram - Failure scenario [1]

and when a failure can occur in the system. The *Failure* element represents a failure domain and specifies the location of an error. The metamodel distinguishes between software (*SWFailure*), hardware (*HWFailure*) and network failure (*LinkFailure*) domains. The *Domain* element represents the consequences of a failure. It defines the failure types *Timing*, *Content* and in addition to [2] *Crash*. The *Timing* and *Content* elements specify failure types that do not cause immediate system interruption but affect the system otherwise. The *Timing* element's attributes *delay* and *scalingFactor* model time delays either due to fixed disturbances or the extension of certain execution times. The *Content* element specifies the circumstances affecting system behavior by giving a corruption value. A system failure can occur later as a result. The attribute *degreeOfCorruption* specifies how the return values, related parameters or variables of the system should deviate from correct values. The *Crash* element represents a component failure that interrupts the execution without any detailed description. By default, the metamodel considers an error as *permanent*. A concrete *FailureType* can now be modelled by creating a class which derives from both *Failure* and *Domain*. For instance, a *HWCrashFailure* as applied in the context of this work, describes a hardware failure in the form of a server failure. For the complete list of *FailureTypes* see [11].

FailureScenario: A *FailureScenario* integrates failures into the overall system model by a chronological sequence of the failure occurrences. The *Occurrence* references a dedicated failure type via the *Failure* element and specifies its temporal occurrence by the attribute *pointInTime*. Thus the same failure type can occur multiple times in the system at different point in times. The *Reference* element

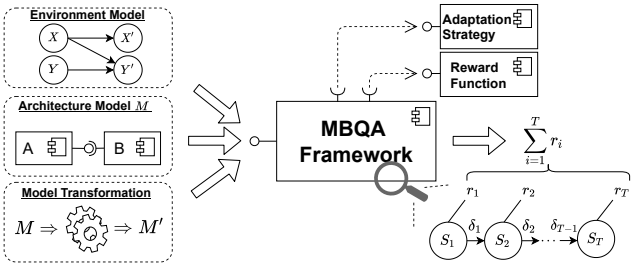

Figure 4: SimExp - Overview of the MBQA framework (adopted from [18]).

determines the failure type's manifestation, i.e. the failure's origin in the system.

Failure scenario simulation: A failure event represents a specific failure type that occurs at a distinct point in time of the simulation. The *FailureScenario* model specifies which failure type should occur when at system simulation time. The *FailureType* model and the *FailureScenario* model together thus form the base for SimuLizar to create all required failure events of the simulation. SimuLizar builds for each *Occurrence* in all *FailureScenarios* a failure event. In SimuLizar the concept of *Behavior* encapsulates the impact of an failure event to the system behavior. For each concrete *FailureType* SimuLizar calculates the impact to the system behavior. An interpreter in SimuLizar uses an *Behavior* to change the simulated resources which has an impact on the simulation result.

A more detailed discussion about the internals of SimuLizar's failure scenario support is out of scope due to limited space.

4.5 Analysis framework integration

We enhanced the MBQA framework SimExp [17] (see figure 4 for overview) to support the DT evaluation of fault-tolerant runtime adaptation strategies in uncertain situations, integrating the concepts discussed in subsections 4.1-4.4.

The SimExp framework runs a simulation of the system over time based on the following inputs: (i) environment model, (ii) architectural model and (iii) a set of model transformations. The environment model captures the operating environment and can be considered as a probability distribution to characterize the stochastic nature of the environment. The architecture model based on the PCM metamodel represents the system architecture [16]. The set of model transformations represent the adaptations to the architecture model. Applying a model transformation evolves the current architecture model to a new architecture model that corresponds to the system after applying an adaption at runtime. The simulation is driven by sampling environmental states from the environment model and by applying model transformations whenever the adaptation strategy decides to adapt the system. The simulation procedure implements Monte Carlo methods to simulate an MDP expressed as Dynamic Bayesian Network (DBN, see [17] for details). The resulting accumulated reward serves as a basis to evaluate design decisions within a family of strategies or to compare two different strategies.

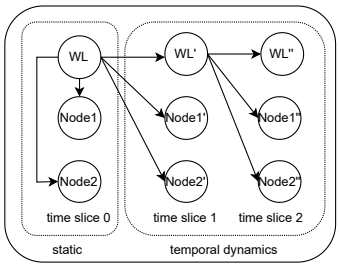

Figure 5: Modeling of node failures - DBN of environment model unrolled over 3 steps

4.5.1 Failure representation - Environment model. Following classic performability literature [12], our approach represents the object system as the architectural model and the operating environment as the environment model. The environment model contains events, that are observable in the system's environment and have an impact on the system's quality attributes. In case of performability these are the uncertainty factors discussed in 4.2. Our approach only considers failure events which have an impact on the system's structure. We extend the environment model with a domain-specific view for failure representation by supplying a persistent failure model. A failure event transitions the system to a faulty state. **Modeling of failure events** is done on the type level as environment variables in the environment model. We unified different kinds of failure types as defined by [2]. All failures are represented as failure events. Our assumptions were: A failure may represent a failed node. A node represents a resource container with a single CPU. A node can be in one of two states [available, unavailable]. Failures are stochastically independent but may depend on the workload. Each failure is modelled as a random variable following a multinomial distribution. By using a DBN [17] we can model the environmental state at time t as a tuple of random variables $(WL, N_1, \ldots, N_n)$ in which WL specifies the probability of the current workload and N_i the probability of a resource failure of server node i. This is achieved by modeling an directed acyclic graph containing a set of nodes, i.e. the random variables $(WL, N_1, \ldots, N_n)$ and a set of edges to describe the dependencies or correlations between the random variables. For example, there is one directed edge each from WL to N_1 and N_2 to indicate that an increased workload correlates with observing a specific node failure. The DBN models the temporal dynamics of node failure occurrences by inductively specifying their probabilistic evolution from time t to $t+1$ (i.e. $P(WL'_{t+1}, N'_{1_{t+1}}, \ldots, N'_{n_{t+1}} \mid WL_t)$) which can be unrolled (by sampling from the distribution) for many time steps. Figure 5 shows our modeling scenario of hardware failures defined as DBN unrolled over 3 steps.

4.5.2 Reward calculation - Performability metric based reward function. Performability evaluation uses various performance indicators. In our context, we have chosen the RT as system performance indicator. However, how do system failures manifest themselves in the RT? From a theoretical perspective, the RT is infinite in case of failures. From a practical perspective we argue that in a real system requests can be dropped in case of failures. This yields in an exceptional behavior that should be handled accordingly by the system. Consequently the RT will not fully reflect the unexpected behavior. Thus we adjusted the traditional performability metric of summing up the weighted RTs by introducing the success rate (SR) as additional performance indicator. The applied performance simulation tool SimuLizar [3] predicts the system's RT as the time span between a measured start and endpoint of a system call. The failure occurrence has no impact on the RT and is not reflected in a declined measured RT. To determine the occurrence of failures during a simulation requires an additional metric to calculate the SR of a system call. Both metrics together permit a statement about the manifested impact of failures on the system's RT. They serve as our performability-based metric for reward calculation. The performability-based reward function r is:

$$r : S \times A \times S \rightarrow [0, 2], r(s, a, s') = \alpha \cdot rt(s') + \beta \cdot sr(s') \quad (1)$$

$rt(s')$ and $sr(s')$ represent the simulation procedures to predict RT and SR. The weights α and β encode preferences for when one performance indicator is preferred over the other. The rest of the paper defines $\alpha = \beta = 1$.

4.5.3 Performance simulation with failures - Model transformation. We simulate node failures by using the SimExp framework [17] and SimuLizar [3] together. The SimExp framework provides connectivity to various analysis tools through model transformations. In our case of performance simulation with failures we provide a model transformation from the SimExp framework's architectural and environmental models to SimuLizar's *FailureScenario* model.

Classic performability evaluation models the stochastic nature of static systems as MRMs (see section 3.1) whereas a SAS is captured by an MDP which is an extension of MRMs. The SimExp framework simulates the SAS's stochastic nature by sampling trajectories over time from the sample space. A sampled state consists of the environment state and the architectural configuration (see section 3.2). For each time step t, the SimExp framework performs the following steps: (1) vector creation consisting of the random variable values of WL and all node states $Node_i$. (2) transformation of the sampled SAS state into an analytical model. This analytical model is then analyzed by running a SimuLizar performance analysis with failures. **Failure models** enable the support of failure simulation in SimuLizar (see subsection 4.4). This work considers node failures as hardware failures. Although SimuLizar supports the occurrence of failure events at any point in time, in our case the simulation is started with the occurred failure event. Thus we model a node failure as a *HW crash failure* which shall be triggered at point in time $t = 0$ in the simulation. **Model to model transformations (M2M)** are used to transform the state (or rather the models associated with a state, i.e. architecture and environment model) to the failure model which is simulated to predict quality attributes. The environment model's random variable values of node states $Node_i$ are mapped to hardware crash failures in the failure model. A sampled failure is represented as *HW crash failure* type per resource container which occurs in SimuLizar at simulation start time t=0.

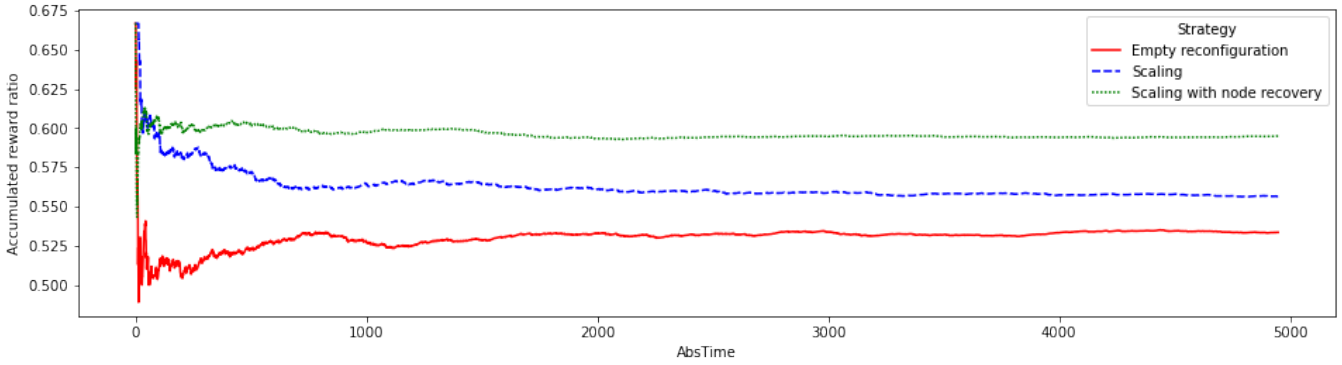

Figure 6: Total accumulated reward of strategies

5 PROOF OF CONCEPT

To demonstrate its applicability we implemented a prototype. The main objective is to investigate whether it supports software engineers in developing fault-tolerant runtime adaptation strategies. This servers as a starting point for a more comprehensive evaluation to determine and evaluate the effectiveness of adaptation strategies with respect to QA fulfillment in case of runtime uncertainties.

Case Study: We used the *Znn.com* system based load balancer. The architecture model is modelled with PCM [3]. In our scenario both server nodes are subject to node failure with an assumed probability of 0.9 (available) and 0.1 (unavailable). We created an environment model based on [17]. The node failures of both servers and the workload as random variables form a DBN (see figure 5). We complemented the resulting DBN with conditional probability distributions (CPDs) that describe the probabilistic evolution of the node failures and workloads that an adaptation strategy must respond to. For the proof-of-concept, we assumed the distributions for each CPD.

Setup: We investigated three adaptation strategies: (i) $\pi_\emptyset$ without any adaptation thus reflecting the behaviour of a static system. (ii) scaling π_S that re-distributes load by adapting the distribution factors (iii) scaling with node recovery π_{SR} that re-distributes load while considering two factors: detected RT threshold violations as described by strategy π_S and detected node failures. The reward function returns a reward $r \in [0, 2]$ per sampled state based on the measured and normalized RT and SR.

Discussion: We evaluated the strategies by sampling 50 trajectories each of length 100 yielding a total of $T = 5000$ sampled states. To compare the strategies, we calculated for each strategy the ratio of the current accumulated reward to the best possible accumulated reward by $ratio_\pi(t) = \frac{1}{2t} \cdot \sum_{i=0}^{t} r_i$. The results (see Figure 6) show the convergence behavior of the accumulated reward ratios towards a fixed ratio with increasing number of samples. Thus the reward ratios and $\overline{SSP}$ induce the following ordering: $ratio_{\pi_\emptyset}(T) < ratio_{\pi_S}(T) < ratio_{\pi_{SR}}(T)$ which indicates the quality of each strategy in terms of meeting the performability-specific quality objectives, π_{SR} performs best and $\pi_\emptyset$ worst.

6 RELATED WORK

Meyer defines the classic performability metric as the expectation value of performance under the condition that things may fail [12]. Trivedi et al. calculate the performability metric by using the weighted sum of performance per system state for static systems [19]. Haverkoort et al. provide a framework for performability modeling and evaluation [9]. Grassi et al. propose the usage of MRMs for dynamically reconfigurable systems [8]. All authors investigated performability in respect of static software systems. Model-based techniques for DT analyses of SASs like SimuLizar [3] simulate adaptation strategies but do not allow the evaluation of their long-term effects. Moreno et al. [14] propose an impact model specification supporting the impact analysis of adaptation effects to help selecting the best corrective action. Camara et al. [4] presents a probabilistic model checking approach for offline synthesis of adaptation strategies applicable at DT to enable a preliminary selection of potentially best strategy candidates. Both approaches are not applicable in performability scenarios as they do not consider performability-specific analysis and metrics.

7 CONCLUSION AND FUTURE WORK

We presented a model-based approach to evaluate performability-oriented runtime adaptation strategies at DT. This work together with the work of [17] form the basis for the presented vision of DT performability optimization of runtime adaptation strategies [15]. Our contribution combines established performability concepts with an MBQA evaluation framework for runtime adaptation strategies to support DT analysis of performability-oriented use cases. This enables: (i) evaluation of DT decisions of runtime artefacts and (ii) knowledge gained from DT analysis applicable at runtime as initial system configuration. We demonstrated the applicability of our approach with a proof-of-concept. We plan to evaluate our approach with a case study to show the accuracy of DT results compared to runtime results.

ACKNOWLEDGMENT

This publication is partly based on the research project SofDCar (19S21002), which is funded by the German Federal Ministry for Economic Affairs and Climate Action. We would like to thank our

colleague Sebastian Krach who contributed significant parts of the *Failure* model by supervising the bachelor thesis [11].

REFERENCES

[1] Palladio-addons failurescenario. https://github.com/PalladioSimulator/Palladio-Addons-FailureScenario (2021 (accessed December 11, 2022))

[2] Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing **1**(1), 11–33 (jan 2004). https://doi.org/10.1109/tdsc.2004.2

[3] Becker, M.W.: Engineering self-adaptive systems with simulation-based performance prediction. Ph.D. thesis (2017). https://doi.org/10.17619/UNIPB/1-133

[4] Cámara, J., Schmerl, B., Moreno, G.A., Garlan, D.: MOSAICO: offline synthesis of adaptation strategy repertoires with flexible trade-offs **25**(3), 595–626. https://doi.org/10.1007/s10515-018-0234-9

[5] Cheng, S.W., Garlan, D., Schmerl, B.: Evaluating the effectiveness of the rainbow self-adaptive system. In: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. IEEE (2009). https://doi.org/10.1109/seams.2009.5069082

[6] Esfahani, N., Malek, S.: Uncertainty in self-adaptive software systems. In: Software Engineering for Self-Adaptive Systems II, pp. 214–238. Springer (2013)

[7] Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer **37**(10), 46–54 (oct 2004). https://doi.org/10.1109/mc.2004.175

[8] Grassi, V., Mirandola, R., Sabetta, A.: A model-driven approach to performability analysis of dynamically reconfigurable component-based systems. In: Proceedings of the 6th international workshop on Software and performance - WOSP '07. ACM Press (2007). https://doi.org/10.1145/1216993.1217011

[9] Haverkort, B.R.: Performability modelling : techniques and tools. Wiley, Chichester [u.a.] (2001)

[10] Howard, R.A.: Dynamic probabilistic systems, vol. 2: Semi-Markov and decision processes. Wiley, New York [u.a.] (1971)

[11] Lehmann, J.: Modeling and Simulating Dependent Failure Scenarios using Palladio. Bachelor's thesis, Karlsruhe Institute of Technology (KIT), Karlsruhe (2021)

[12] Meyer, J.F.: On evaluating the performability of degradable computing systems. IEEE Transactions on Computers **C-29**(8), 720–731 (aug 1980). https://doi.org/10.1109/tc.1980.1675654

[13] Moreno, G.A., Cámara, J., Garlan, D., Schmerl, B.: Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In: Proceedings of the 2015 10th joint meeting on foundations of software engineering. pp. 1–12 (2015)

[14] Moreno, J.C., Lopes, A., Garlan, D., Schmerl, B.: Impact models for architecture-based self-adaptive systems. In: Formal Aspects of Component Software. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-15317-9_6

[15] Rapp, M., Scheerer, M., Reussner, R.: Design-time performability optimization of runtime adaptation strategies. In: Companion of the 2022 ACM/SPEC International Conference on Performance Engineering. p. 113–120. ICPE '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3491204.3527471

[16] Reussner, R.H., Becker, S., Happe, J., Heinrich, R., Koziolek, A., Koziolek, H., Kramer, M., Krogmann, K.: Modeling and Simulating Software Architectures – The Palladio Approach. MIT Press, Cambridge, MA (10 2016)

[17] Scheerer, M., Rapp, M., Reussner, R.: Design-time validation of runtime reconfiguration strategies: An environmental-driven approach. In: 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). IEEE (aug 2020). https://doi.org/10.1109/acsos49614.2020.00028

[18] Scheerer, M., Reussner, R.: Reliability prediction of self-adaptive systems managing uncertain ai black-box components. In: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). IEEE (2021)

[19] Trivedi, K.S.: Probability and Statistics with Reliability, Queuing and Computer Science Applications. John Wiley & Sons, Inc., 2nd edn. (sep 2016). https://doi.org/10.1002/9781119285441

[20] Weyns, D.: An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective. WILEY IEEE COMPUTER SOC PR (2020)