computers

MDPI

Article

Evaluation of Self-Healing Systems: An Analysis of the State-of-the-Art and Required Improvements

Sona Ghahremani * and Holger Giese

Hasso Plattner Institute, University of Potsdam, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany; holger.giese@hpi.de

* Correspondence: sona.ghahremani@hpi.de

Received: 12 January 2020; Accepted: 14 February 2020; Published: 27 February 2020

Abstract: Evaluating the performance of self-adaptive systems is challenging due to their interactions with often highly dynamic environments. In the specific case of self-healing systems, the performance evaluations of self-healing approaches and their parameter tuning rely on the considered characteristics of failure occurrences and the resulting interactions with the self-healing actions. In this paper, we first study the state-of-the-art for evaluating the performances of self-healing systems by means of a systematic literature review. We provide a classification of different input types for such systems and analyse the limitations of each input type. A main finding is that the employed inputs are often not sophisticated regarding the considered characteristics for failure occurrences. To further study the impact of the identified limitations, we present experiments demonstrating that wrong assumptions regarding the characteristics of the failure occurrences can result in large performance prediction errors, disadvantageous design-time decisions concerning the selection of alternative self-healing approaches, and disadvantageous deployment-time decisions concerning parameter tuning. Furthermore, the experiments indicate that employing multiple alternative input characteristics can help with reducing the risk of premature disadvantageous design-time decisions.

Keywords: self-healing; failure model; performance; simulation; evaluation

1. Introduction

A self-adaptive system (SAS) is capable of modifying itself at runtime in response to the changes of the environment and the system itself. Violations of certain functional and nonfunctional goals trigger the self-adaptation [1]. Equipping the software system (adaptable software) with an external adaptation engine, such as a MAPE-K feedback loop, enables the realization of self-adaptive capabilities. The evaluation of self-adaptive systems is not trivial. On one hand, these systems often have a complex structure due to an additional control layer. The system is also subject to changes in unforeseen ways as a result of the adaptation [2]. On the other hand, these systems are designed to be operated in highly dynamic environments, and therefore require continuous monitoring of their behavior and execution environment [3].

A self-healing system (SHS) can discover and diagnose the runtime disruptions such as failures and react to them by dynamically adapting and reconfiguring the system. Self-healing systems are usually characterized by restrictions (e.g., adaptation is only needed if failures occur) [4]. In the specific case of SHS, the evaluation of the performance of the alternative self-healing approaches and their parameter

tuning relies on the considered characteristics of failure occurrences and the interaction between the occurrence of failures and the self-healing approach.

Over the last few decades, a huge body of work on engineering SAS including SHS has been developed by researchers and engineers. As distinguished in [3], during the lifetime of a SAS in general, evaluation concerns design-time, deployment-time, and runtime decisions. As a first step in evaluating a SAS, the quality of the employed self-adaptive approaches is investigated. The choice of the proper adaptation approach is a design-time decision. The quality of the design-time decisions significantly influences the overall quality attributes of SAS [5,6]. Performance and optimality are the main quality attributes meaningful for evaluation of SAS. The studies focusing on evaluation at this level are concerned with early stage performance of the system during the development process.

The second group of evaluation attempts for SAS are concerned with investigating the performance achieved through self-adaptivity. This group does not directly study the self-adaptation approaches, but monitors the SAS at deployment-time or runtime in its fully operational mode and analyses the quality of adaptation after each adaptation cycle. This step requires exposing the SAS to a wide spectrum of the potential inputs and investigating the quality of its behavior. Runtime and deployment-time adjustments to the self-adaptation approaches such as parameter tuning are subject to this class of evaluation.

The two groups of the studies on evaluation of SAS require sound and reproducible experiments on the system, under controlled and customizable conditions. Simulators allow evaluating the system under controlled and reproducible conditions. This enables the comparison of multiple solutions under similar circumstances.

For the evaluation of SHS, the possibility to inject failures is critical for simulated experiments [7]. The reason is that failures are rare events, and therefore testing for them in the operational environment requires long measurements and is often not feasible. Moreover, testing for failures in the operational environment is often also not desirable due to the possibly negative consequences. However, using simulators comes with the problem that the simulated model and its characteristics always only mimic the behavior of the real system to a limited extent, and therefore conclusions that are made based on simulation may lack generality [8].

In this paper we at first study the state-of-the-art for evaluating the performance of SHS by means of a classification of the approaches concerning their considered input and a systematic literature review, both of which refine our findings presented in [9]. Furthermore, we extend the considerations of [9], by also analysing the identified classes of approaches employing specific input types and discussing their limitations. Our systematic literature review conducted in [9] and extended in this work reveals that while majority of the existing work on performance evaluations of SHS use simulation-based evaluation schemes (97%), often the characteristics considered for the occurrences of failures in these studies are not sophisticated, and thus the validity of the outcome is often not clear. Workload related metrics such as response time, throughput, and working versus adaptivity time need to be measured to analyze the performance for SAS [10].

The input for a SHS is a trace of failure occurrences (i.e., failure trace). Employing customizable failure traces with statistical parameters provides a workload with controllable volatility for SHS. Only employing such input traces (i.e., failure traces) can enable thorough and credible evaluation of SHS (see [11]).

We further study the impact of the identified limitation concerning the characteristics of the occurrences of failures via four hypotheses. The proposed hypotheses indicate that: (I) Incorrect assumptions regarding the characteristics of the input trace can result in large prediction errors for the performances of self-healing approaches. (II) Employing the wrong characteristics for the occurrence of failures can result in disadvantageous design-time decisions concerning the selection of alternative self-healing approaches and (III) disadvantageous deployment-time decisions concerning parameter

tuning. Finally, (IV) employing multiple alternative input traces with volatile characteristics of the occurrence of failures can reduce the risk of premature disadvantageous design-time decisions.

The case study employed throughout this paper is mRUBiS [12], an instance of the common RUBiS that is frequently used for validating self-adaptation mechanisms [13]. mRUBiS is an online marketplace hosting an arbitrary number of shops. Each shop contains 18 components of different types. Components of a shop can adapt and be configured independently of each other. Self-healing capabilities are added to mRUBiS via running a MAPE-K feedback control loop. We investigate the proposed hypotheses through a set of empirical experiments on mRUBiS. We explore a set of self-healing approaches in combination with multiple failure traces to demonstrate the validity of the hypotheses via several test cases.

The empirical assessments of the proposed hypotheses confirm that inaccurate assumptions regarding the characteristics of the input failure trace of a SHS at design-time can result in up to 138% performance prediction error of the self-healing approaches and up to 51% performance loss, as they might result in disadvantageous and premature choices of self-healing approaches. The experiments further demonstrate that tuning the optimization parameter of the self-healing approaches based on incorrect assumptions regarding the characteristic of the input trace at deployment-time can cause up to 575% performance loss for the SHS. Finally, the empirical assessment of the hypotheses suggests that employing multiple input traces to steer the choice of self-healing approaches at design-time reduces the risk of premature wrong decisions by 48%.

This paper extends our previous work [9] with the following novel contributions: (I) We extend the study of the state-of-the-art for evaluating the performance of SHS by providing a classification of different input types for simulated SHS. (II) For each recognized input category for a simulated SHS, we analytically discuss the validity scope and limitations of the generated output. (III) We demonstrate the potential impacts of insufficient considerations of the input trace characteristics on the performance of SHS and the risk of making disadvantageous premature decisions via proposing and empirically validating four hypotheses.

The paper is structured as follows: The problem space of evaluating SHS together with a classification of the input types for simulated SHS is discussed in Section 2. Section 3 presents the state-of-the-art on the performance evaluation for SHS. In Section 4 we analytically discuss the legitimacy of the claims regarding the evaluation of SHS. We also discuss for each input type of SHS, the feasibility of different output types. Section 5 presents four hypotheses regarding the impact and importance of proper consideration of input characteristics for performance evaluation of SHS. The section provides an empirical validation of the hypotheses on several variations of simulated SHS together with the threats to the validity of the results. A discussion of the findings together with suggestions for required improvements in evaluation of SHS is presented in Section 6. Section 7 concludes the paper and provides an outlook for the planned future work.

2. On the Evaluation of SHS

A SHS is a reactive system that responds to external stimuli or failures (i.e., input) with repair actions in form of system reconfigurations. In the dependability literature, such component failures are named faults, while only failures of the overall system are named failures (see [14]). For simplicity, we employ here the terminology of [11] where this distinction is not made. In the following we describe the problem space for evaluation of SHS and provide a classification of potential input types for simulated SHS.

2.1. Problem Space of SHS Evaluation

Figure 1 extends the generic diagram of the system under evaluation presented in [15] and sketches the three main elements of the SHS under evaluation: (1) the *input* of the SHS, (2) the adaptable system together with the adaptation engine forming the *system under evaluation*, and (3) the *output* of the SHS.

There are four elements shaping the problem space of the SHS and its evaluation [16]: *failure model*, *system response*, *system completeness*, and *design context*.

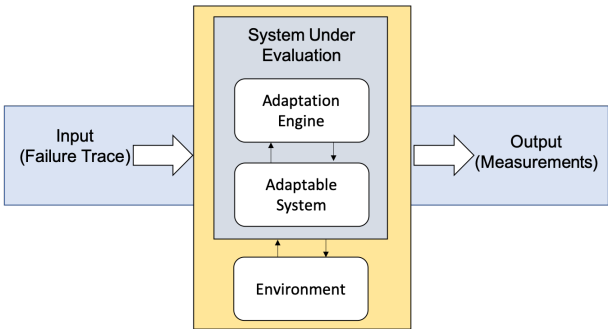

Figure 1. Elements of SHS under evaluation.

The *failure model* captures the hypotheses concerning the relevant failures, their characteristics, and their distribution. This corresponds to *input* (i.e., failure trace) in Figure 1. *System response* (*output* in Figure 1) captures the ability of the system for fault detection, the response of the system to the detected faults, and the correctness assurance for the system response (see [16]). *System completeness* reflects the completeness of the system architecture, evolution, self-knowledge, and designer knowledge. *Design context* refers to other factors that influence the scope of self-healing, such as user involvement, system scope, abstraction level, and behavioral predetermination.

The *failure model* and *system response* play substantial roles in particular for evaluation of SHS [17]. To evaluate the output or *system response* of SHS via simulated experiments, it is required that the simulator reflects properly whether a chosen repair activity leads to the expected effects. System performance is reported to be one of the most common quality attributes that is measured as system response for evaluation of SHS and SAS in general [5,6]. The performance of a system can be realized via various metrics capturing the quality of the service provided by the system [18]. Robustness claims for the performance measurements, or in general any output measurements of a system under evaluation are only justified if the system is evaluated in the presence of various volatile input conditions [19].

For a SHS under evaluation, a representative input refers to a (set of) failure trace(s) with volatile characteristics thoroughly describing properties of the potential operational environment of the SHS. Input of a SHS under evaluation is considered to be reliable if it prevents disadvantageous decisions at design-time or deployment-time that are purely made based on the assumptions regarding the characteristics of the input.

To achieve a proper *failure model*, the simulator is required to faithfully reflect both the characteristics of the failures and of the failure occurrences (i.e., failure profile). Defining a *failure model* is necessary for any SHS [7]. Koopman [16] outlines characteristics such as failure occurrence duration, failure manifestation, failure source, granularity, and failure profile expectations—the elements that should be captured by a failure model for a SHS.

As studies of failure occurrences observed for real systems indicate, failures are often not independently occurring but correlated in time or space (referred to as failure bursts) [11]. A failure profile can be characterized via the attributes such as *failure group size*, *inter arrival time*, and *failure exposure time*, as shown in Figure 2.

Failure group size (FGS) is the number of time or space correlated failures that occur approximately at the same time or within a short time span. *Inter arrival time (IAT)* is also known as mean time between failures (MTBF). *IAT* represents the time between two occurrences of failure groups or bursts. *Failure*

exposure time (FET) is the time window during which time or space correlated failures affect the system. Thus, each burst occurs within the FET.

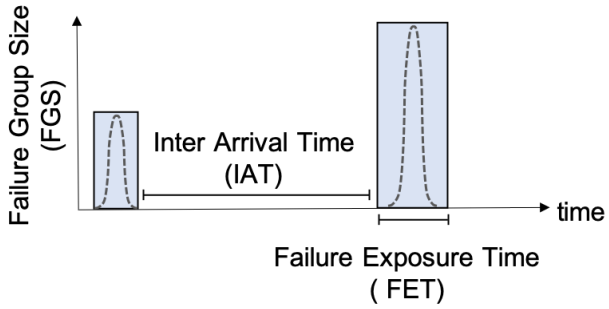

Figure 2. Characteristics of a failure profile.

The *granularity* and the *magnitude* of the failures are also enumerated as features that shape a failure model [16]. To obtain a failure profile, intensive studies of the considered (simulated) system over a considerably long period (months or years) are required.

2.2. Classification of Input Types for Simulated SHS

A SHS under evaluation can be either simulated or executed. The execution of a SHS is either in its planned operational environment or in an artificially synthesized one. When operating the system in its planned operational context, it is often not possible to observe all the relevant system characteristics. The environment characteristics also cannot be influenced. Furthermore, certain changes such as component failures directly affect the operational system, and therefore, are costly. Consequently, the community usually employs simulators to mimic the environment and sometimes even the characteristics of the SHS [6].

Figure 3 presents a classification of the possible input types for simulated SHS. The tree structure depicts the identified types ranging from the most naive on the left to the most complex on the right side of the tree (both outlined with dash lines). As stated in [14], the construction of the test inputs (i.e., *synthetic failure model*) for a simulated system can be either *deterministic* or *probabilistic*. A *deterministic failure model* defines the characteristics of the failure profile with scalar values and therefore, generates traces with deterministic characteristics for occurrences of failures. A *probabilistic failure model* employs probability distributions to characterize a failure profile. Such a statistically defined failure model is either an outcome of fitting statistical models to recordings of real data (*fitted to real data*) or does not fit parts (or any) of the failure profile attributes to real data (i.e., *not fitted to real data*).

A probabilistic failure model that is *not fitted to real data* is constructed based on artificially synthesized probability distributions. Characteristics of failure occurrences such as FGS, IAT, and FET are defined based on probability distributions. However, the employed distributions are not based on real data. Random or uniform distribution of failures are examples of such probabilistic traces.

A probabilistic failure model which is *fitted to real data* characterizes the occurrence of the failures with respect to multiple attributes as shown in Figure 2. The failure occurrence characteristics are obtained via fitting statistical distributions to real data. As a result, traces generated from these failure models are considered more representative of real world failure traces compared to the alternatives, which are either probabilistic traces *not fitted to real data* or *deterministic traces*.

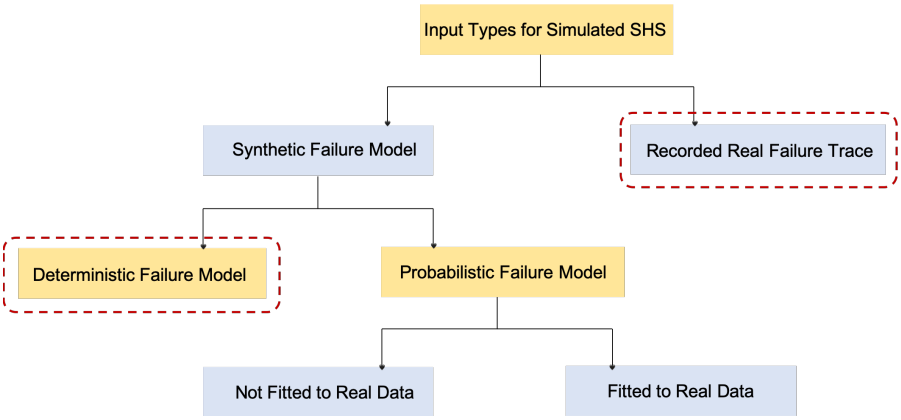

Figure 3. Classification of input types for a simulated SHS.

Between the *probabilistic failure models not fitted to real data* and ones that are fully *fitted to real data* there is a wide range. The spectrum starts from failure models that are fully artificial and none of the failure occurrence characteristics are extracted from or fitted to real data. In between are the traces whose characteristics are partially manually defined and partially fitted to real data. An example is a failure model which fits the failure density (i.e., the overall number of the failures in the trace) and FGS to real data but the values for IAT are defined manually. On the other end of the spectrum are models that fit all the possible failure occurrence characteristics to real data.

Finally, a *recorded real failure trace* in Figure 3 refers to a continuous recording of failure occurrences in a real system. The trace captures the realistic characteristics of the failure occurrences. A simulated experiment can be steered via injecting failures accordingly by replaying the trace.

3. State-of-the-Art

A systematic literature review (SLR) on the state-of-the-art in evaluating the performance of SHS is presented in [9]. This section complements the findings of [9] by mapping the results of the conducted SLR to the classification of the input types for simulated SHS presented in Section 2.2. To that end, the classification of the investigated studies in [9] is modified accordingly to be aligned with the improved classification of the SHS input types proposed in this paper. The conducted research method in this part follows the standard practice in systematic literature reviews [20].

3.1. Research Questions

This literature review attempts to answer the two research questions (*RQ*):
RQ1: What is the state-of-the-art in evaluating the performance of SHS?
RQ2: How are the failure traces designed for the evaluation of SHS?

3.2. Selection Method

The adopted scheme to search the literature and the selection criteria for the analyzed studies are specified in the following.

3.2.1. Search Term and Query String

We aim to identify studies in the context of SAS, focusing on self-healing behavior. To this end, we used the term “self-healing” as the query string. To make sure that all the relevant studies on SHS are covered, we extend the search to the metadata (i.e., title, abstract, and keywords).

3.2.2. Searched Databases and Venues

To study the state-of-the-art in evaluating the performance of SHS, we investigated the papers published in the main conferences and journals in the areas of autonomic, self-adaptive, self-organizing, and self-managing systems. Therefore the searched venues are the TAAS journal and the conferences ICAC, SASO, and SEAMS along with their companion workshops. We carried out automatic searches on the ACM DL and IEEE Xplore databases. We set no limit on the publication year, even though the concepts of autonomic computing and self-* systems have developed only since 2002.

3.2.3. Inclusion and Exclusion Criteria

The scope of our literature review is restricted to performance evaluation of SHS. Table 1 enumerates the considered inclusion criteria (IC) and exclusion criteria (EC) along with the number of the studies satisfying each criterion. After retrieving the preliminary search results from the explored databases, each paper is analyzed against the IC and EC listed in Table 1. A paper is considered in our literature review if it satisfies all IC and no EC.

Table 1. Inclusion and exclusion criteria.

Inclusion Criteria	**# of Studies**
IC-1: There is an implementation for a SHS	41
IC-2: The performance of SHS is measured	39
Exclusion Criteria	
EC-1: There is no implementation of a SHS	36
EC-2: The performance of SHS is not measured	2
EC-3: A more complete version of the study is selected	3
Total included studies (satisfying all IC and no EC)	36

3.3. Selected Studies

The search on the databases was conducted on 1 March 2019. The preliminary search resulted in a total number of 77 studies. Table 2 shows the distribution of the search results among the considered venues before and after applying the IC and EC.

Table 2. List of venues and search results.

Venue	**Initial # of Studies**	**# of Studies after Applying IC and EC**
ICAC & Workshops	36	14
SASO & Workshops	20	7
SEAMS	19	14
TAAS	2	1
Sum of tudies	**77**	**36**

Each study is evaluated against the IC and EC listed in Table 1. In summary, 36 studies satisfy all IC and no EC, which are included in the literature review.

3.4. Results

For our literature review, we selected 36 papers that provide a performance evaluation for SHS. In the following we reflect on the analyzed studies to answer the research questions.

3.4.1. RQ1

RQ1: What is the state-of-the-art in evaluating the performance of SHS?

97% of the analyzed studies employ simulators to mimic a real SHS in their performance evaluations. We identified a single study (3%) that executes a real system to evaluate the performance of SHS [21]. This is inline with the related finding of [6], which suggests that simulation-based experiments are the most dominant approach employed for the evaluation of SAS in general.

3.4.2. RQ2

RQ2: How are the failure traces designed for the evaluation of SHS?

The selected studies are further analyszed regarding their choices on designing the input for SHS.

Figure 4 summarizes the design choices of inputs for the evaluation of SHS in the reviewed studies. Our SLR revealed eight studies (22%) that do not report on the employed trace for the occurrences of failures in their evaluations of SHS [7,22–28]. Therefore, we only have data for 78% of the investigated papers. A large proportion of the reviewed papers, 16 studies (44%), employ single *deterministic traces* to steer the failure injection in the simulated experiments [17,29–43]. In these studies the characteristics of the failure occurrences such as FGS and IAT are not determined by a probability distribution but determined for each occurrence explicitly as scalar values (see Section 2.2). FET is not recognized as a dimension for characterizing the employed failure traces.

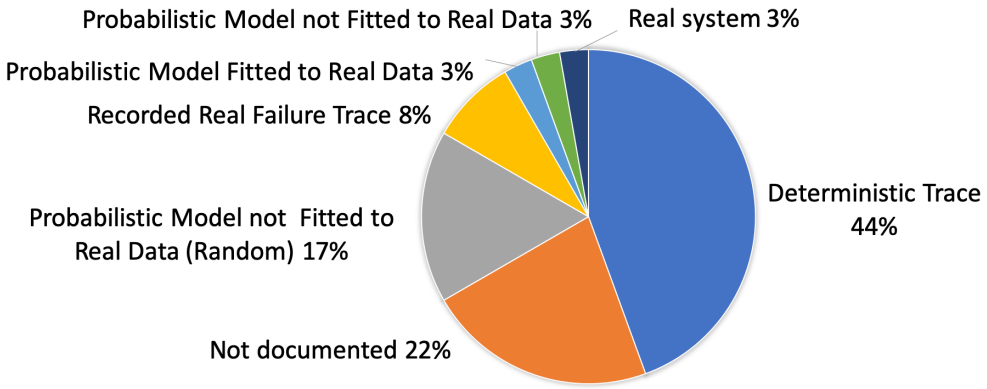

Figure 4. Choice of input for SHS among the studies.

Six studies (17%) use a naive variation of *probabilistic failure models not fitted to real data*, which is *random* occurrences of failures to characterize the input for the SHS [44–49]. In these studies, the number of the failures (i.e., FGS) is deterministic, but the *IAT* and *FET* are randomly defined. We identified three studies (8%) that use *recorded real failure traces* as input for simulated SHS. In [50–52], manually adjusted recordings of the real-world failure traces are injected to the simulated SHS. However, a representative model of the nature of the source trace including *FGS*, *IAT*, and *FET* of the failure occurrences is missing from [50–52]. The *FGS*, *IAT*, and *FET* of the failure occurrences are cleary defined via statistical distributions in [53,54]. A *probabilistic failure model not fitted to real data* is used in [53] to generate multiple failure traces, and [54] employs multiple failure traces generated by *probabilistic failure model fitted to real data*.

Our literature review identifies a single paper using a real system for the evaluation of SHS [21]. In this paper, failures are not artificially injected to the system and occur as the real system equipped with self-healing properties is executed in its real operational environment. This scheme for the evaluation results in long experiments since the failures occur infrequently. The experiments in [21] however lack

reproducibility and controllability, since they are not conducted under controlled conditions, and the feasibly of reproducing the exact failure traces is not investigated.

4. Analytical Assessment

In the following we analytically assess the scope of credibility and trustworthiness of SHS evaluation for each of the input types classified in Figure 3. For each recognized input type, we discuss which claims regarding the input and output of the SHS are justified and whether the evidence suffices for qualitative and (or) quantitative evaluation of the output.

4.1. Deterministic Failure Model

A *deterministic failure model* and the resulting deterministic traces do not aim to be generic and only provide minimal coverage of the input space for SHS, and hence fail to suffice for a quantitative evaluation of the SHS. As discussed in Section 2.1, a representative input space for SHS includes employing multiple arbitrarily long failure traces with volatile characteristics regarding failure occurrences. Employing multiple *deterministic traces* does not improve the evidence regarding generality and robustness, since they cannot provide more than only partial coverage of the SHS input space. The coverage of the input space is limited to the variants of the considered deterministic traces and is not extendible. Therefore, such traces are not representative.

Deterministic traces support qualitative evaluation only for the considered fraction of the input space and do not suffice to address generic claims on quantitative evaluation of the SHS. However, multiple arbitrary long deterministic traces with volatile characteristics can support qualitative evaluation for the segments of the input space covered by the traces. On the other hand, multiple deterministic traces with volatile characteristics cannot be considered reliable to steer the design-time and deployment-time decisions, because they only describe limited instances of the possible futures for a SHS under evaluation.

4.2. Probabilistic Failure Model

Probabilistic failure models employ statistical distributions to define the characteristics of failure occurrences, allowing the generation of multiple arbitrarily long failure traces with desired characteristics. This provides a reasonably good coverage of the potential input space of the SHS and supports qualitative evaluation of the SHS. As a result, employing *probabilistic failure models* to generate input traces for a SHS under evaluation allows robustness testing for performance measurements and can be considered reliable to steer the design-time and deployment-time decisions.

As discussed in Section 2, *probabilistic failure models* vary from *not fitted to real data* to those which are fully *fitted to real data*. As the choice of input for SHS changes from *probabilistic failure models not fitted to real data* towards the ones *fitted to real data*, the failure models and resulting failure traces are more representative of the real operational environment of the SHS under evaluation. This improves the credibility of the quantitative evaluation of the simulated SHS.

A *probabilistic failure model* that is fully *fitted to real data* obtains *all* the possible characteristics of the failure occurrences from fitting statistical distributions to real data. The quality of such models and resulting traces rely on the amount and the quality of the real data used to derive the probabilistic distributions. In addition, if the *probabilistic failure model* is *not fully fitted to real data*, the extent and the type of the manually defined characteristics of the failure occurrences influence the quality and credibility of the input.

Any *probabilistic failure model* that manually defines parts or all of the failure occurrence characteristics is not fully representative of the potential spectrum of the SHS input space, because certain characteristics are either not captured at all or are not necessarily representative of real failure traces. Therefore,

probabilistic failure models not fully fitted to real data fail to support any claims of being representative of a real operational environment for a SHS which is key for the credibility of the quantitative analysis of the simulation-based outputs.

A *probabilistic failure model fitted to real data* on the other hand, as discussed in Section 2, fully captures the characteristics of the failure occurrences and is representative of the real operational environment for the SHS. Employing multiple traces generated from a *probabilistic failure model fitted to real data* as input supports credible quantitative and qualitative evaluation of the SHS, since all the characteristics of failure occurrences are represented via statistical distributions and can be tuned to cover a large spectrum of the input space for SHS.

4.3. Recorded Real Failure Trace

While playing back a recording of a real failure trace as input for a simulated system improves the credibility of the simulation-based experiments, the resulting output of a single *recorded real failure trace* lacks generality. Employing a single *recorded real failure trace* as input for SHS evaluation only supports single experiment run and does not support any claim on certain qualitative evaluation metrics, such as resilience, reliability, and robustness testing [3]. While such a trace contains realistic characteristics of occurrences of failures, it only captures one possible future for the simulated SHS and fails to cover a large and representative spectrum of the input space. We argue that in this case the output of the SHS under evaluation is inconclusive and cannot support any qualitative evaluation on the performance, since it lacks generality. Therefore, multiple *recorded real failure traces* with different characteristics which evaluate the performance of the SHS for multiple future operation contexts are required. This supports the required number of the experiments to obtain robust results.

To collect multiple representative recordings of real traces, the real system needs to be monitored for a considerably long time to identify all the correlations between the failures. Therefore, since failures are often rare events and do not occur frequently, having access to multiple *recorded real failure traces* for a simulated system is often not feasible or very costly to obtain.

Among all the possible variations of the input types for evaluation of simulated SHS (see Figure 3), multiple *recorded real failure traces* or multiple failure traces generated from *probabilistic failure models* which are fully *fitted to real data* can provide a representative input set for evaluation of the SHS that can be tested for robustness of the results. A representative *recorded real failure trace* or a *probabilistic failure model* which is *fitted to real data* allow deriving simulated behavior which mimics the characteristics of failure occurrences in a real system. Therefore, when conducting simulated experiments for SHS, only the two input types would allow justifying general claims regarding certain quality attributes including performance, throughput, robustness, etc. Any other input type for the simulated SHS does not result in conclusive output for the purpose of quantitative evaluation. Qualitative evaluation of SHS can be supported if the employed failure traces(s) provide good coverage of the input space of SHS. However, as discussed earlier in Section 2, the effort to obtain multiple recordings of real systems can be very high or even infeasible. A trade-off solution can be employing *probabilistic failure models* which are fully *fitted to real data*.

5. Empirical Assessment

In this section we present four hypotheses on the importance of input considerations for design, deployment, and evaluation of SHS. We validate the proposed hypotheses via a set of empirical experiments.

5.1. Case Study

The empirical experiments are conducted employing a simulator of mRUBiS—an online marketplace that hosts an arbitrary number of shops, each consisting of 18 components [12]. Each shop can be configured differently and runs isolated from the other shops. mRUBiS is a variant of the common RUBiS that is frequently used for testing and validating self-adaptation mechanisms [13].

We are particularly interested in *self-healing*, i.e., to automatically repair runtime failures; therefore, we equip mRUBiS with a MAPE-K feedback loop (adaptation engine in Figure 1). The simulator is also equipped with failure injection capabilities; hence, it is suitable for injecting failures according to different distributions. The simulator emulates failures in mRUBiS which are detected later on via the self-healing properties enabled by the MAPE-K feedback loop. The variant of the applied mRUBiS hosts 100 shops. Overall, the simulator includes 1800 components.

5.2. Measurements: Utility and Reward

As a measure of performance for the self-healing approaches, we consider the accumulated utility of the SHS over time (i.e., reward) [55]. To compute the utility values for the SHS, a utility function is required to express how well each configuration of the system in its domain satisfies the functional and non-functional goals of the system [56]. Then, the utility can be accumulated over time to determine the obtained reward.

In general, for a software system, utility can be obtained from service level agreements (SLA), learned, or analytically constructed [54]. In our experiments we employ the analytical scheme proposed in [43,57] to determine the utility of dynamic software architectures. However, the specific choice of the utility function elicitation method is not important here, as it is the same for all the considered alternative self-healing approaches in this study. The employed utility function evaluates the system configuration after each adaptation (i.e., repair of runtime failures). The utility and the reward of the system are measured independently from the employed self-healing approaches.

In mRUBiS—our employed case study—the overall utility of a shop is the sum of the sub-utilities of all the components in the shop. The utility of the system is the sum of the utilities of all shops.

5.3. Spectrum of Considered Self-Healing Approaches

Self-healing approaches can be categorized in various ways. In our empirical experiments we structure the spectrum of the considered self-healing approaches with respect to their employed planning schemes. The spectrum is limited to three self-healing approaches developed in our former work [43,58]:

Static. In this approach, the cost and the benefit of the repair actions are estimated at design-time. Based on these static estimates, the repair actions for each type of failures are assigned statically. The drop of the system utility caused by each type of failure is estimated at design-time, which leads to a fixed order in which the failure types are addressed by this approach. Since the approach is purely static, it does not add any runtime overhead due to planning for the repair of the failures (see [43,58]).

Solver. This approach employs a costly runtime optimization to maximize the overall utility of the system during the planning phase. Since the approach uses a constrain solver to optimize the objective function, we refer to it as the solver approach. The approach uses the utility function as the objective function to form an ordered sequence of repair actions. The tasks of assigning proper repair action to each failure and prioritizing them for execution are defined as optimization problems. As shown in [58], the approach is an optimal (concerning system utility) but expensive heuristic employing optimization-based planning, and suffers from a large runtime overhead.

U-driven. This is an optimal and cost efficient approach employing utility-driven planning. The impacts of different repair actions are computed at runtime by employing a utility function. Following

greedy decision making, the repair actions resulting in the largest impacts on the overall utility are chosen to be applied first. The order in which failures are resolved and the proper repair actions to resolve them are decided based on the runtime observations regarding the affected components and the utility drops caused by the failures.

5.4. Execution Horizon

The *execution horizon* is an ordered list of planned repair actions chosen for execution during the current self-healing loop. The size of the execution horizon, k, is a parameter that is relevant for model predictive control [59], sequence planning [60], and the planning phases of various approaches (e.g., [61]) that employ scheduling strategies for the potential adaptation steps. Therefore, inspired by the model predictive control, we impose the execution horizon as an optimization parameter of the self-healing approaches which affects the performance of the SHS.

The *planing horizon* is a list of all the repair actions assigned to all the observed failures before the current self-healing loop. During the planning phase of a self-healing loop, the self-healing approach assigns the proper repair actions to the detected failures. The assignment and ordering of the repair actions is steered based on design-time or runtime estimates of the impact of the repair actions (see Section 5.3).

An execution horizon of size k, however, only considers the first k repair actions in the planning horizon for execution in the current self-healing loop (see Figure 5). After repairing the k corresponding failures, the current self-healing loop ends and the remaining unresolved failures are considered together with the newly occurred failures in a subsequent self-healing loop. For example, if the execution horizon is of size one (i.e., $k = 1$), only the first planned repair action is executed and the re-planning is initiated in the subsequent self-healing loop based on the new observations and remaining unresolved failures.

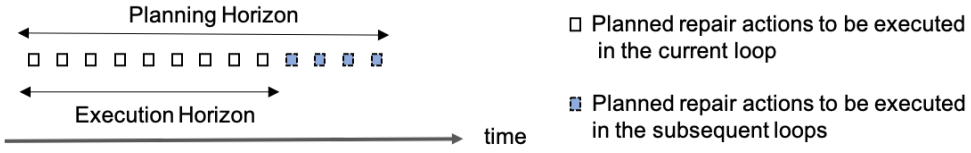

Figure 5. Planing and execution horizon.

The advantage of execution horizon of size one (or small sizes in general) is that the self-healing approach utilizes the most recent failures immediately, as opposed to ignoring them until the execution of all the planned repair actions ends. The disadvantage of applying a small size for the execution horizon is that the repair of several failures (that are in the planning horizon and not in the execution horizon) is delayed to the subsequent loop(s). This delays the potential improvements of the utility as a result of the repair action executions. In cases where the planning phases of the self-healing approaches have large runtime overheads (e.g., the solver approach in Section 5.3), the resulting delay in the respective utility improvements can be considerably large and cause sever performance loss for the SHS.

5.5. Selected Traces for Failure Occurrences

As revealed by the foundational work on characterizations of failure occurrences in computer systems [62–64], failures often have a *bursty* character. Studies of failure traces observed for real systems indicate that failures are often not occurring independently, but correlated in time or space (referred to as failure bursts) [11]. This phenomenon occurs due to failure propagation in the system where the occurrence of a single failure triggers a sequence of failures in other parts of the system within a short time span. Numerous fault tolerant algorithms [65,66] make the strong assumption that failures occur independently.

However, bursty failure traces do not act in accordance with this assumption. The occurrence of failure bursts often makes the availability behavior of different system components correlated. Ignoring the bursty character of the failure occurrences results in overestimating the transient reward rate by an order of magnitude [67]. This is the case even when only as few as 10% of the failures conform to a bursty distribution.

To investigate the impact of the input characterization on the performance of SHS, we employed several failure traces with volatile characteristics covering a large spectrum of the potential input space of a SHS. In the employed traces, the characteristics of the failure occurrences are described via IAT, FGS, and FET. In failure traces where the distribution of the failures within each burst is not characterized, we assume a failure propagation following a normal distribution, as shown in Figure 2. Following our discussions in Section 4, we limit the scope of the employed input types to *probabilistic failure models*. We expose the simulated SHS to variants of failure traces generated from *probabilistic failure models* which are either fully *fitted to real data* or partially fit the characteristics of the failure occurrences to real data.

5.5.1. Probabilistic Failure Model Fitted to Real Data

In the following we describe the Grid'5000 failure model, one of the probabilistic failure models presented in [58] which is fully fitted to real data and employed in the experiments of this study.

Grid'5000 is an experimental grid environment of over 2500 processors with 1288 nodes [68]. The data recorded for Grid'5000 failure model are gathered over 1.5 years of monitoring [69]. Gallet et al. [11] provide statistical distributions for different characteristics of a failure model fitted to the collected data of the grid environment. Table 3 lists the distributions proposed by [11] for IAT and FGS along with the considered FET.

Based on the probabilistic distributions for IAT and FGS and the values for FET, we can derive a failure trace(s) with an arbitrary length. For the experiments, we generated the failure trace Grid'5000 with $n = 50$ failure bursts spread over 24 h. As depicted in Table 3, the overall number of the failures (i.e., failure density) in the generated Grid'5000 trace is 1116 (specifically, failures which affected the SHS under evaluation within 24 h). Figure 6 shows the distribution of the FGS values in the generated Grid'5000 trace.

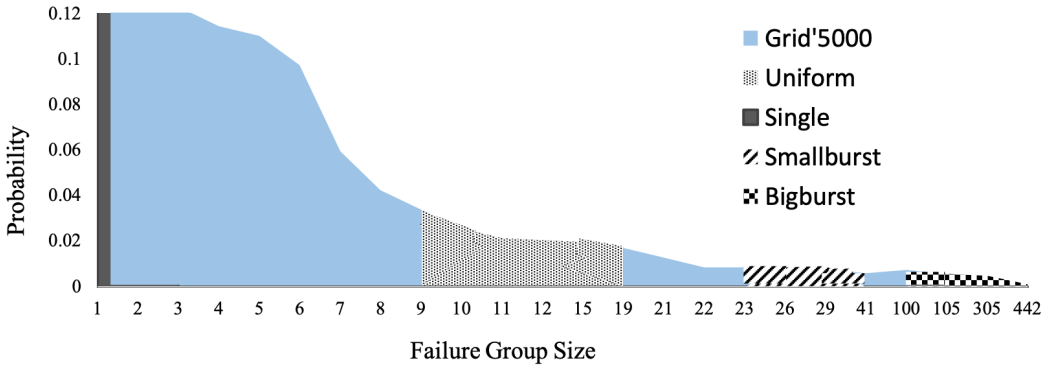

Figure 6. Failure group size (FGS) distribution in Grid'5000 trace and considered areas for extracted probabilistic failure models.

Table 3. Characteristics of the employed probabilistic failure models.

	Grid'5000	**Uniform**	**Single**	**Smallburst**	**Bigburst**
FGS	$LOGN(1.88, 1.25)$	$N(15.15, 5.63)$	1	$N(32.45, 7.41)$	$N(238, 97.3)$
IAT (s)	$LOGN(-1.39, 1.03)$	864	77.4	$N(1112.4, 500.61)$	$N(3521.4, 5418.6)$
FET (s)	250	50	N/A	100	250
Number of Bursts	50	100	1116	75	6
Duration (h)	24	24	24	24	24
Failure Density	1116	1116	1116	1116	1116

5.5.2. Probabilistic Failure Models Partially Fitted to Real Data

We manually modified the parameters of the Grid'5000 failure model provided by [11] to obtain failure models with volatile characteristics. The resulting failure models are intended to provide a wide and representative coverage of the potential input space for the SHS. They also allow us to study the impact of extreme characteristics of failure occurrences on the performance of SHS.

Uniform, Single, and Bigburst failure models are the three probabilistic failure models partially fitted to real data presented in [58] which are also considered in the experiments of this study. We extend the set in this paper by adding one more failure model to the set, referred to as Smallburst. In the following we describe the characteristics of the employed probabilistic failure models which are partially fitted to real data.

Uniform is a probabilistic failure model employed to generate a trace of failure bursts uniformly distributed within 24 h. In the Uniform model, the failure density and FET are fitted to real data, since the model preserves the failure density and FET of the original Grid'5000 failure model. The FGS and IAT are defined as following:

Using statistical bootstrapping, a normal distribution is extracted from the set of FGS values (referred to as set F) in the Grid'5000 trace [70,71]. Then, set F is randomly re-sampled to multiple sample sets. F' is the set comprising the mean values of each sample set. A normal distribution $N(\mu_{F'}, 2\sigma_{F'})$ is then used to generate FGS values for the Uniform trace. The outcome is a set of uniformly distributed values within a certain margin extracted from set F. Applying this distribution, a sequence of normally distributed values for FGS within the desired margin is generated.

The number of the bursts in the Uniform trace is double the number of the bursts in the Grid'5000 trace (i.e., 50×2), since the average of the FGS in the Uniform failure model is approximately 50% smaller than in the Grid'5000 failure model. Therefore, in the Uniform trace, the overall number of the failures occurring within 24 h (i.e., 1116) is distributed among 100 bursts. Figure 6 sketches the portion of the original FGS values from the Grid'5000 trace that are present in the Uniform trace.

The IAT of the occurrences of failures in the Uniform failure model is the average of IAT values in the Grid'5000 failure model. As a result, the employed Uniform trace is a sequence of bursts with normally distributed FGS values which occur in equal intervals. Table 3 presents the characteristics of the Uniform failure model.

Single is a failure model generating failure traces where failures are not correlated and arrive individually, not in groups. Our SLR in Section 3 revealed that most of the existing work investigating the performance of SHS employ naive failure traces similar to traces generated by the Single failure model (see [29,31–33,35,72]).

In the generated Single failure trace for our experiments, individual failures are randomly distributed within the 24 h. The Single trace preserves the failure density of the original Grid'5000 model. In order to obtain the Single trace, the number of the failure groups is exactly equal to the failure density, 1116, since

each failure group includes exactly *one* failure (i.e., FGS = 1), as shown in Figure 6. Table 3 presents the characteristics of the Single failure model.

Smallburst is a failure model representing correlated arrival of failures (i.e., bursts), but the sizes of the bursts are scaled down compared to the original Grid'5000 failure model. To construct the Smallburst failure model, we employed statistical bootstrapping on traces generated from the statistical distributions of Grid'5000 presented in Table 3. As depicted in Figure 6, the targeted FGS in this failure model is distributed approximately between 25 to 40 failures during each burst. To this end, only portions of the generated Grid'5000 trace with FGSs between 25 and 40 are re-sampled for statistical bootstrapping.

Similarly to the previous failure models, the Smallburst failure model also preserves the failure density of the original Grid'5000 failure model. The FGS values in the Smallburst failure model are on average smaller than the ones in the Grid'5000 failure model. Therefore, to obtain the same failure density, the number of the bursts in the generated Grid'5000 trace (i.e., 50) and the IAT values decrease accordingly. IAT values for the Smallburst failure model are extracted via bootstrapping from the randomly re-sampled IAT values of the generated Grid'5000 trace and then adjusted manually to sum up to 24 h. The characteristics of the Smallburst failure model are depicted in Table 3.

Bigburst failure model represents the other end of the spectrum of failure occurrence characteristics (compared to the Single failure model) and generates only large failure bursts. Similar to the scheme employed for Uniform and Smallburst failure model, to construct the Bigburst failure model, statistical bootstrapping of the probabilistic distributions in Grid'5000 is applied. To obtain large FGS values, only the part of the set F (i.e., FGS values in the generated Grid'5000 trace) above a certain threshold (FGS $\geq$ 100) is re-sampled for bootstrapping. This is depicted in Figure 6.

The Bigburst failure model also preserves the failure density of the Grid'5000 failure model. To keep the same failure density, since the Bigburst failure model only includes large bursts, the number of the bursts decreases respectively compared to the Grid'5000 failure model. This results in an increase of the IAT values for the Bigburst failure model. IAT values for the Bigburst failure model are extracted via bootstrapping from the randomly re-sampled large IAT values of the Grid'5000 trace (IAT $\geq$ 1000 *sec*). The characteristics of the Bigburst failure model are depicted in Table 3.

5.6. Hypotheses and Validation

In the following we introduce four hypotheses to be empirically assessed. The objective of the hypotheses is to steer the evaluation of SHS towards making the right decisions both at deployment-time and design-time. As discussed in Section 1, through these hypotheses we investigate the potential impacts of design-time decisions (choice of the self-healing approaches) and deployment-time decisions (tuning execution horizon size) on the performance of SHS.

We show that wrong assumptions concerning the characteristics of the input trace could steer the design-time and deployment-time decisions towards the incorrect direction and eventually result in sever performance loss. Our empirical assessments of the hypotheses confirm that while the magnitude of the impacts could be considerably large, such effects can only be captured in the presence of *multiple* input traces covering a vast spectrum of failure occurrence characteristics.

Hypothesis 1. *The choice of the input has an impact on the predicted performance of SHS. Wrong assumptions regarding the characteristics of the input trace can result in large performance prediction errors for the self-healing approaches.*

Hypothesis 2. *The choice of the self-healing approach has an impact on the performance of SHS. Wrong assumptions regarding the characteristics of the input trace can result in disadvantageous choice of the employed self-healing approach at design-time, and thus can cause sever performance loss.*

Hypothesis 3. *Tuning the optimization parameter of the parameterized self-healing approaches (e.g., size of the execution horizon) has an impact on the performance of SHS. Wrong assumptions regarding the characteristics of the input trace can result in disadvantageous choice of the employed execution horizon size for the self-healing approach at deployment-time, and thus can cause sever performance loss.*

Hypothesis 4. *In cases where no accurate information about the real operation environment of the SHS is available, employing multiple input traces to steer the choice of self-healing approaches at design-time reduces the risk of premature wrong decisions.*

5.6.1. Validation of Hypotheses

In the following, we conduct a set of experiments to empirically validate the proposed hypotheses. We acknowledge that in this study we investigate the validity of the hypotheses only by employing the available single case study, failure traces, and self-healing approaches. Thus, we can only employ these elements to demonstrate the existence of evidence supporting the proposed hypotheses, but we cannot demonstrate that this applies to all or the majority of cases.

As discussed in Section 5.2, we consider the accumulated reward (i.e., utility over time) as a measure of performance for the SHS. For each failure trace generated from the failure models presented in Table 3, we equip mRUBiS (i.e., a simulated SHS introduced in Section 5.1) with one of the three available self-healing approaches (see Section 5.3). The employed traces have equal failure density (i.e., 1116 failures overall) to preserve comparability of the results and describe occurrences of the failures for 24 h (see Section 5.5). However, while the failure traces last for 24 h, the self-healing period in the worst case occupies only 12% of the considered period. For each combination of the available self-healing approach and failure trace, the self-healing period indicates the time each approach requires to repair all the detected failures. Our measurements reveal that the solver approach presents longer self-healing periods on average, as it often suffers from large runtime overhead due to its costly optimization-based planning (see Section 5.3). The measurements are the average of 1000 simulation runs for each trace-approach combination (The experiments and simulations have been conducted on a machine with OS X 10.14, Intel processor 2.6 GHz core *i*5, and 8 GB of memory.).

5.6.2. Validating Hypothesis 1

To investigate Hypothesis 1, we study the impact of the employed input on the performance of SHS. Table 4 presents the accumulated reward values obtained by each of the self-healing approaches in the presence of the employed failure traces during 24 h.

Table 4. Reward values of self-healing approaches for different failure traces in 24 h.

Failure Trace	Reward		
	U-Driven	**Solver**	**Static**
Single	1.99×10^9	1.99×10^9	1.64×10^9
Uniform	1.95×10^9	1.95×10^9	1.60×10^9
Smallburst	1.93×10^9	1.79×10^9	1.65×10^9
Grid'5000	1.96×10^9	1.53×10^9	1.68×10^9
Bigburst	1.77×10^9	8.35×10^8	1.70×10^9

A representative failure trace supports accurate assumptions regarding the characteristics of the SHS input in its operational environment. This results in realistic predictions for the performance

of self-healing approaches. These predictions can potentially steer the design-time choice of the self-healing approach.

Hypothesis 1 implies that in cases where the employed trace is not representative of the SHS input space (leading to wrong assumptions regarding the characteristics of the input trace), it can result in performance prediction error. To validate Hypothesis 1, using Equation (1), we compute the prediction error of the self-healing approaches in case of wrong assumptions regarding the characteristics of the input trace. As we are only interested in the magnitude of the error and not whether the prediction is too optimistic or pessimistic, we present the absolute values of the prediction error in Equation (1).

$$Prediction\ Error := \frac{|Predicted\ Value - Real\ Value|}{Real\ Value} \tag{1}$$

Tables 5–7 depict the performance prediction errors for the u-driven, solver, and static approaches in the presence of different input traces with equal failure densities (see Section 5.5). As indicated by the results, the solver approach presents the largest sensitivity to wrong assumptions regarding the characteristics of the input trace. The performance prediction errors for the u-driven approach vary from 2% to 12%. The larger error values of the u-driven approach are observed for the case when the runtime (i.e., real) trace is the Bigburst trace and the assumptions suggest different input traces for the predictions. Similarly, in the solver approach, the large prediction errors are also observed for the Bigburst trace. The error values for the solver approach are shown to be as high as 138%. Among the three approaches, the static approach presents less sensitivity to the wrong assumptions regarding the input trace compared to the other two approaches. The performance prediction errors for the static approach vary from 1% to 6%, where the larger values relate to the Bigburst trace both as the runtime trace and as the trace used for predictions. Note that in Equation (1), values above 100% for the *Prediction Error* indicate that the difference between the *Real Value* and the *Predicted Value* is more than twice the *Real Value*.

Table 5. Performance prediction error of u-driven approach in presence of different failure traces.

Failure Trace Used for Prediction	**Runtime Trace**				
	Single	**Uniform**	**Smallburst**	**Grid'5000**	**Bigburst**
Single	-	2%	3%	2%	12%
Uniform	2%	-	1%	1%	10%
Smallburst	3%	1%	-	2%	9%
Grid'5000	2%	1%	2%	-	11%
Bigburst	11%	9%	8%	10%	-

Table 6. Performance prediction error of solver approach in presence of different failure traces.

Failure Trace Used for Prediction	**Runtime Trace**				
	Single	**Uniform**	**Smallburst**	**Grid'5000**	**Bigburst**
Single	-	2%	11%	30%	138%
Uniform	2%	-	9%	27%	134%
Smallburst	10%	8%	-	17%	114%
Grid'5000	23%	22%	15%	-	83%
Bigburst	58%	57%	53%	45%	-

Table 7. Performance prediction error of static approach in presence of different failure traces.

Failure Trace Used for Prediction	Runtime Trace				
	Single	**Uniform**	**Smallburst**	**Grid'5000**	**Bigburst**
Single	-	3%	1%	2%	4%
Uniform	2%	-	3%	5%	6%
Smallburst	1%	3%	-	2%	3%
Grid'5000	2%	5%	2%	-	1%
Bigburst	4%	6%	3%	1%	-

5.6.3. Validating Hypothesis 2

Hypothesis 2 indicates that inaccurate assumptions regarding the input trace characteristics can result in an incorrect choice of the self-healing approach, and thus loss of performance. In the following we investigate the extent of the performance loss as a result of wrong assumptions about the characteristics of the input trace. The u-driven approach is excluded from this set of experiments. As shown in our previous works [43,58] and confirmed by the results in Table 4, the performance of the u-driven approach is robust against the changes of the input trace characteristics due to its incremental planning scheme.

To investigate Hypothesis 2, first we identify the best performing approach (excluding u-driven) for the available input traces. For each input trace, the best performing approach is the one that obtains the largest reward. As depicted in Table 4, for input traces with smaller FGSs (i.e., Single, Uniform and Smallburst) the solver approach obtains higher reward values compared to the static approach. While for traces with larger FGSs (i.e., Grid'5000 and Bigburst) the static approach performs better than the solver approach, since it does not introduce any runtime overhead (see [43]).

The next step to validate Hypothesis 2 is to study the reward loss due to a wrong choice of the self-healing approach at design-time. The reward loss of each self-healing approach in the presence of an input trace is computed according to Equation (2).

$$Reward\ Loss := \frac{Reward - Reward_{best}}{Reward_{best}} \tag{2}$$

For a given input trace, $Reward_{best}$ in Equation (2) represents the achieved reward by the best performing approach for that trace. The *Reward* values for each input trace-approach pair are extracted from Table 4. For each input trace, if the selected approach is the best performing approach, then *Reward Loss* equals zero.

Table 8 presents the *Reward Loss* values for different input traces in case of wrong choices for the employed self-healing approaches at design-time. The design-time decisions in Table 8 are represented as ($Trace_i$, $Approach_j$). This indicates that at design-time, $Approach_j$ is chosen to be employed on the SHS, since the input trace is assumed to be $Trace_i$. If these assumptions regarding the characteristics of the input are incorrect, i.e., the runtime trace has different characteristics than $Trace_i$, then the design-time choice of the self-healing approach does not best fit the runtime trace and results in reward loss.

Table 8. *Reward Loss* of self-healing approaches in case of wrong design-time decisions .

Design-Time Decision	(Runtime Trace, Best Performing Approach)				
	(Single, Solver)	**(Uniform, Solver)**	**(Smallburst, Solver)**	**(Grid'5000, Static)**	**(Bigburst, Static)**
(Single, Solver)	0	0	0	−9%	−51%
(Uniform, Solver)	0	0	0	−9%	−51%
(Smallburst, Solver)	0	0	0	−9%	−51%
(Grid'5000, Static)	−18%	−19%	−8%	0	0
(Bigburst , Static)	−18%	−19%	−8%	0	0

The results presented in Table 8 reveal that if the choice of the self-healing approach is made based on unrepresentative input traces, it could lead to considerably large performance loss (up to 51% in our empirical experiments). As stated in Section 2, reliable input for a SHS under evaluation prevents disadvantageous decisions at design-time or deployment-time. The results confirm that the characteristics of the input trace influence the performance of SHS, and therefore should steer the choice for the employed self-healing approach.

5.6.4. Validating Hypothesis 3

Hypothesis 3 comprises of two parts. The first part of the hypothesis indicates that tuning the optimization parameter of the parameterized self-healing approaches has an impact on the performance of SHS. The second part of the hypothesis states that wrong assumptions regarding the characteristics of the input trace can result in wrong choice of the employed execution horizon size and therefore, can cause sever performance loss.

To validate Hypothesis 3, first (I) we investigate the impact of the execution horizon on the performance of SHS. Then, (II) we study the extent of the performance loss caused by the wrong assumptions regarding the characteristics of the input trace.

I. Impact of the Execution Horizon Size on the Performance of SHS

As discussed in Section 5.4, the size of the employed execution horizon represents an optimization parameter for the self-healing approaches. This value can be tuned at runtime or deployment-time with respect to the characteristics of the input trace and the properties of the self-healing approach (e.g., planning time, etc.).

To investigate the impact of the execution horizon on the performance of SHS, we run each of the self-healing approaches introduced in Section 5.3 with different sizes for the execution horizon in presence of the input traces presented in Section 5.5. The Single input trace is excluded from the experiments due to its naive characteristics. The trace describes only single occurrences of failures, therefore, the notion of an execution horizon is not applicable.

The variant of the self-healing approach without employing any execution horizon is considered as the *baseline* approach. For each combination of input trace-approach, the baseline reward ($Reward_{baseline}$) is the reward achieved by the baseline approach. For each variant of the input trace-approach pair, we study the impact of the execution horizon size in relation to the $Reward_{baseline}$ as shown in Equation (3).

$$Reward\ Delta := Reward - Reward_{baseline} \tag{3}$$

Reward Delta in Equation (3) is a positive value if applying the execution horizon improves the reward compared to the baseline; otherwise, it is a negative value and indicates performance drop.

For all the combinations of the input trace-approach, values higher than 40 for execution horizon size do not result in any change in the observed pattern. Therefore, we only report the *Reward Delta* values for execution horizon sizes up to 40. In the conducted experiments, the size of the employed execution horizon is chosen at deployment-time and does not change at runtime.

Figure 7 presents *Reward Delta* values of the u-driven approach with different execution horizon sizes in the presence of the four input traces. For each input trace, the best performing variant of the u-driven approach (i.e., best execution horizon size) is the one with highest *Reward Delta* value and is hachured in each chart of Figure 7. As the results in Figure 7 present, different input traces with different characteristics result in different best performing values for the execution horizon size. The results for *Reward Delta* of u-driven approach show that as the FGS of the input traces increases, the size of the best performing execution horizon increases as well. The u-driven approach equipped with smaller execution horizon

sizes (i.e., 1 and 2) performs better in presence of the input traces with smaller FGS (i.e., Uniform and Smallburst) while for the input trace with larger FGS (i.e., Bigburst) larger execution horizon size (i.e., 8) performs better.

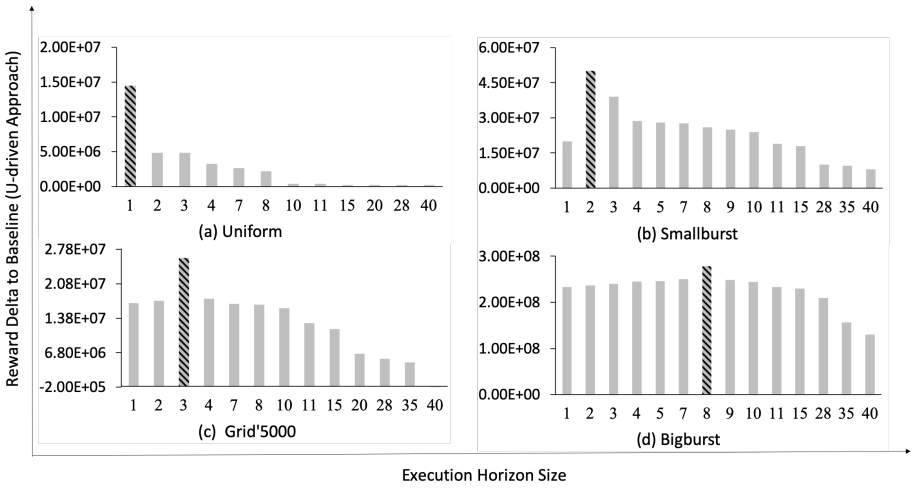

Figure 7. *Reward Delta* values of u-driven approach for different execution horizon sizes.

Figures 8 and 9 show the *Reward Delta* values for the solver and static approaches with different execution horizon sizes in presence of the input traces. Similar to the u-driven approach, for both solver and static approaches, the increase in the FGS of the input trace results in larger values for the best performing execution horizon sizes.

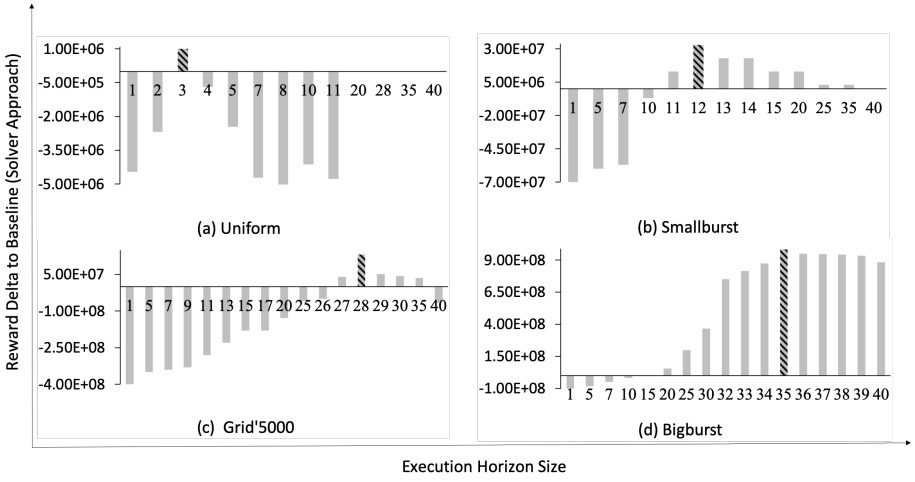

Figure 8. *Reward Delta* values of solver approach for different execution horizon sizes.

The best performing execution horizon sizes for the solver approach are 3 for the Uniform trace, 12 for the Smallburst trace, 28 for the Grid'5000 trace, and 35 for the Bigburst trace. However, for the u-driven approach, as presented in Figure 7, these values are considerably smaller for the same input traces. As shown in our previous work [43], the optimization-based approaches (i.e., the solver approach in this work) employ more costly planning schemes compared to the runtime efficient approaches, such as the u-driven approach. Therefore, while employing such costly approaches, less frequent planning is preferred. In the presence of the large failure group sizes (e.g., Grid'5000 and Bigburst), frequent planning

in approaches with large runtime overheads can bring the system down, as the execution horizon with a small size requires more frequent planning. As depicted in Figure 8 for the Grid'5000 and Bigburst traces, the execution horizons with smaller sizes cause severe drops in the system performance for the solver approach.

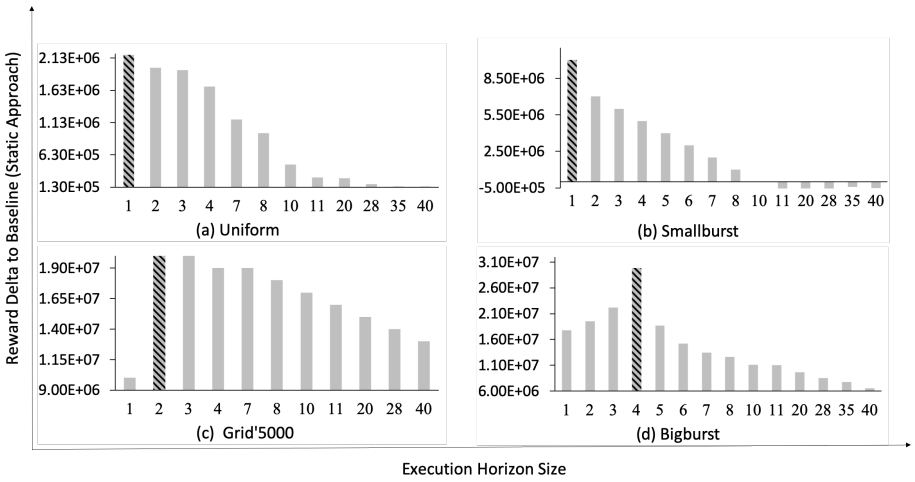

Figure 9. *Reward Delta* values of static approach for different execution horizon sizes.

The *Reward Delta* values for the static approach in Figure 9 are inline with our observation regarding the correlation between the runtime overhead of the approach and the best performing execution horizon size. Since the static approach has no runtime overhead due to its design-time predictions (see Section 5.3), for the same input trace, it requires smaller or the same size for the best-performing execution horizon compared to the u-driven approach which, as shown in [43], introduces a small overhead at runtime.

For the same reason, the differences between the best performing execution horizon sizes are even larger if we compare the static approach (Figure 9) to the solver approach (Figure 8). We show in [43] that the solver approach has a considerably larger runtime overhead compared to the static approach.

The results presented in Figures 7–9, validate the first part of the Hypothesis 3. The performance of the SHS, employing any of the studied self-healing approaches, either with design-time or runtime predictions for planning, is affected by tuning the execution horizon size.

II. Performance Loss of SHS due to the Wrong Choice of Execution Horizon Size

In the following we study the extent of the performance loss caused by the wrong assumptions about the characteristics of the input trace (i.e., second part of Hypothesis 3). To this end, we compute the *Reward Loss* for each approach caused by the wrong deployment-time choices regarding the employed execution horizon size. For each of the u-driven, solver, and static approaches, using Equation (2), *Reward Loss* values are computed and presented in Tables 9–11 respectively. The deployment-time decisions are presented as ($Trace_i$, *execution horizon size* $= j$) pairs. This indicates that at deployment-time, j is chosen as the size of the execution horizon for the self-healing approach, since the input trace is assumed to be $Trace_i$. If the assumptions about the characteristics of the input are incorrect, i.e., the runtime trace has different characteristics than $Trace_i$, then the deployment-time choice of the execution horizon size does not best fit the runtime trace and results in reward loss.

The results presented in Tables 9–11 show that for all the self-healing approaches and a given input trace, if the execution horizon size is tuned based on the wrong assumptions regarding the characteristics of the input trace, the SHS loses reward. For instance, in Table 9, if the input trace (i.e., runtime trace) is

Uniform, while the execution horizon size is tuned based on the assumptions for the Bigburst trace, the SHS will lose 85% of its reward.

Table 9. *Reward Loss* of u-driven approach compared to the best performing execution horizon sizes for different traces.

Deployment-Time Decision	**(Runtime Trace, Best Performing Execution Horizon Size)**			
	(Uniform, 1)	**(Small Burst, 2)**	**(Grid'5000, 3)**	**(Big Burst, 8)**
(Uniform, **1**)	0	−60%	−35%	−16%
(Small Burst, **2**)	−67%	0	−34%	−15%
(Grid'5000, **3**)	−69%	−22%	0	−14%
(Big Burst, **8**)	−85%	−48%	−36%	0

Table 10. *Reward Loss* of solver approach compared to the best performing execution horizon sizes for different traces.

Deployment-Time Decision	**(Runtime Trace, Best Performing Execution Horizon Size)**			
	(Uniform, 3)	**(Smallburst, 12)**	**(Grid'5000, 28)**	**(Bigburst, 35)**
(Uniform, **3**)	0	−297%	−385%	−110%
(Smallburst, **12**)	−575%	0	−291%	−102%
(Grid'5000, **28**)	−100%	−91%	0	−63%
(Bigburst, **35**)	−100%	−91%	−74%	0

Table 11. *Reward Loss* of static approach compared to the best performing execution horizon sizes for different traces.

Deployment-Time Decision	**(Runtime Trace, Best Performing Execution Horizon Size)**			
	(Uniform, 1)	**(Smallburst, 1)**	**(Grid'5000, 2)**	**(Bigburst, 4)**
(Uniform, **1**)	0	0	−50%	−40%
(Smallburst, **1**)	0	0	−50%	−40%
(Grid'5000, **2**)	−10%	−30%	0	−35%
(Bigburst, **4**)	−23%	−51%	−6%	0

Our experiments show that the solver approach is the most sensitive approach to the wrong tuning of the execution horizon. As shown in Table 10, the performance loss of the solver approach can be as high as 575%. While the u-driven and static approach present more robustness against the wrongly tuned execution horizons, the performance loss can be as high as 85% for the u-driven approach and up to 50% for the static approach (see Table 9 and 11).

Our empirical results in this section suggest that wrong choices of the execution horizon size—due to assumptions based on *unreliable* input traces—can result in sever performance loss of the SHS (up to 575%). Therefore, it is essential that the tuning of the optimization parameters of the parameterized self-healing approaches is done with respect to the characteristics of the input trace.

5.6.5. Validating Hypothesis 4

We refer to the set of input traces used at design-time to steer the choice of self-healing approaches as $\mathbb{S}$. The size of $\mathbb{S}$ (i.e., $|\mathbb{S}|$) indicates the number of the employed input traces. Hypothesis 4 indicates that in cases where no accurate information about the real operational environment of the SHS is available, increasing $|\mathbb{S}|$ reduces the risk of making wrong choices of the employed self-healing approach. To validate

Hypothesis 4, we investigate how the risk of making wrong choices about the self-healing approach evolves as $|\mathbb{S}|$ increases.

In our empirical experiments, the upper bound for $|\mathbb{S}|$ is 5 (i.e., the number of the available input traces introduced in Section 5.5). The likelihood of correct, inconclusive, and wrong decisions are calculated for $1 \leq |\mathbb{S}| \leq 5$. Similar to the experiments in Section 5.6.3, the u-driven approach is excluded from the experiments here, since it outperforms the solver and static approaches for all the available input traces (see Table 4). In order to demonstrate Hypothesis 4, we need to have an alternating set of best performing approaches.

A *correct* decision refers to the cases where the self-healing approach suggested by the traces in $\mathbb{S}$ is the best performing approach for the runtime trace as well. The best performing approach (excluding u-riven approach) for each of the five available input traces can be extracted from Table 4. We use the notation $\mathbb{S} \rightarrow approach_j$ if all the traces in $\mathbb{S}$ suggest $approach_j$ to be chosen as the best performing approach. $Runtime\ trace \rightarrow approach_i$ indicates that $approach_i$ is the best performing approach for the runtime trace. For the set of considered self-healing approaches in this section, likelihoods of the correct decisions are calculated as in Equation (4).

$$\begin{aligned} P(Correct) = P(\mathbb{S} \rightarrow Solver) \times P(Runtime\,trace \rightarrow Solver) \\ + P(\mathbb{S} \rightarrow Static) \times P(Runtime\,trace \rightarrow Static) \end{aligned} \tag{4}$$

A *wrong* decision refers to a case where the employed self-healing approach, suggested by $\mathbb{S}$, is not the best performing approach for the runtime trace. The likelihood of the wrong decisions can be computed as in Equation (5) for our set of considered approaches.

$$\begin{aligned} P(Wrong) = P(\mathbb{S} \rightarrow Solver) \times P(Runtime\,trace \rightarrow Static) \\ + P(\mathbb{S} \rightarrow Static) \times P(Runtime\,trace \rightarrow Solver) \end{aligned} \tag{5}$$

An *inconclusive* outcome refers to the cases where the traces in $\mathbb{S}$, for $2 \leq |\mathbb{S}| \leq 5$, do not suggest the same self-healing approach to be employed as the best performing approach. Any case which is not correct or wrong is categorized as an inconclusive outcome. Therefore, the likelihood of inconclusive cases can be computed as in Equation (6). In the conducted experiments we assume that all the traces in $\mathbb{S}$ are equally likely to accurately represent the runtime trace and also are equally likely to be included in $\mathbb{S}$.

$$P(Inconclusive) = 1 - P(Correct) - P(Wrong) \tag{6}$$

Figure 10 presents the likelihoods of the correct, inconclusive, and wrong design-time decisions for $1 \leq |\mathbb{S}| \leq 4$. We excluded the results for $|\mathbb{S}| = 5$, as they are similar to the case $|\mathbb{S}| = 4$. For $|\mathbb{S}| = 1$, where the decision for the choice of self-healing approach is steered by only one input trace, the solver approach is chosen as the best performing approach for the Single, Uniform, and Smallburst traces (see Table 4). Therefore, $P(\mathbb{S} \rightarrow Solver) = 60\%$. For $|\mathbb{S}| = 1$, since the choice of the self-healing approach is made only base on one input trace, $P(Runtime\,trace \rightarrow Solver) = P(\mathbb{S} \rightarrow Solver) = 60\%$. If Grid'5000 or Bigburst trace steer the choice of the self-healing approach at design-time, the static approach is chosen as the best performing approach with 40% likelihood (i.e., $P(\mathbb{S} \rightarrow Static) = 40\%$). In this case also $P(Runtime\,trace \rightarrow Static)$ equals $P(\mathbb{S} \rightarrow Static) = 40\%$. Employing Equations (4)–(6) for $|\mathbb{S}| = 1$ results in 52% likelihood of the correct decisions, while there is a risk of 48% to make wrong decisions. As the case $|\mathbb{S}| = 1$ contains only one trace in the set $\mathbb{S}$, it does not lead to an inconclusive situation, as depicted in Figure 10.

Employing two input traces (i.e., $|\mathbb{S}| = 2$) reduces the likelihood of the wrong decisions to 18%; the likelihood of the correct decisions also drops to 22%; and 60% of the cases are reported as inconclusive. Adding one more trace to $\mathbb{S}$ ($|\mathbb{S}| = 3$) reduces the likelihood of the wrong decisions to 4%. Correct decisions are observed with 6% likelihood, and in 90% of the cases the outcome of the set $\mathbb{S}$ is inconclusive.

For $|\mathbb{S}| = 4$, the likelihoods of both wrong and correct decisions drop to zero, and the only possible outcome is inconclusive (i.e., likelihood of 100%). As elaborated earlier, the inconclusive output for $\mathbb{S}$ indicates that the traces in $\mathbb{S}$ do not suggest the same best-performing self-healing approach. Our experiments show that by increasing $|\mathbb{S}|$, the likelihood of wrong decisions drops to zero. However, increasing $|\mathbb{S}|$ eventually results in only achieving an inconclusive output. The objective behind increasing $|\mathbb{S}|$ is to reduce the risk of making wrong decisions, and our experiments confirm this claim. In addition, we argue that inconclusive output for $\mathbb{S}$ is also beneficial, as it prevents wrong choices.

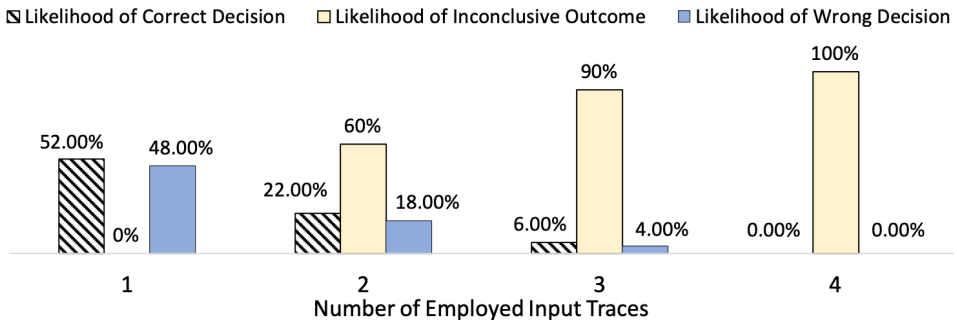

Figure 10. Likelihood of design-time decisions in relation to the number of employed input traces for evaluation.

We argue that the large likelihood of inconclusive output for large $|\mathbb{S}|$ values is due to the limitations of our empirical experiments and the restricted number of the available input traces. If the input space of the SHS is represented more fine grained with considerably more input traces, the likelihood of inconclusive output for $\mathbb{S}$ can be expected to decrease respectively, since it is more likely that traces with similar characteristics are included in the set $\mathbb{S}$. However, in our experiment for $|\mathbb{S}| = 4$, since traces with extreme differences in their characteristics are included in $\mathbb{S}$ (e.g., Bigburst and Single), it is expected to observe inconclusive outputs.

The results of Figure 10 confirm that as the number of the considered input traces to steer the choice of the self-healing approach (i.e., $|\mathbb{S}|$) increases, the risk of making wrong decisions drops considerably (from 48% to zero in our experiments). In addition, increasing $|\mathbb{S}|$ supports robustness claims for the performance measurements, as it allows multiple measurements of the system under evaluation in presence of various volatile input conditions (see Section 2.1).

5.6.6. Threats to Validity

Internal Validity

Threats to internal validity concern how we performed the experiments and interpreted the results. We rule out any experimentation bias and selection bias, as the set of employed self-healing approaches and failure traces used for the empirical assessments were developed in our former works and independent of the experiments in this paper. Our experiments in [43,58] indicated that the u-driven approach outperforms the solver and static approaches, and the experiments conducted in this study confirm this finding; however, this did not cause any bias in the experiments and the interpretation of the results, as comparing the optimality of the approaches regarding the reward is not the subject of this study. Moreover,

the experiments are conducted and repeated several times using the controlled simulation environment mRUBiS [12] following the benchmark guidelines proposed by [73] to obtain trustworthy measurements and results.

External Validity

Threats to external validity may restrict the generalization of our findings outside the scope of our experiments. Such threats are the use of only one system under evaluation, the specific failure traces, and the three self-healing approaches. To mitigate these threats, we used mRUBiS equipped with self-healing properties as the system under evaluation, which allows the injection of generic failure traces. We can consider mRUBiS as a generic and representative exemplar for self-healing.

The choice of the traces is intended to provide a reasonable representation of the potential spectrum of failure occurrence characteristics. Employing real world traces and traces statistically derived from real data to cover the edge cases and extreme characteristics allowed us to mitigate the threat of generalization of the results. However, certain measurements, such as performance loss are heavily affected by the characteristics of the failure occurrences, and thus the reported measurements cannot be generalized, as they can be considerably lower or higher in the presence of failure traces with different characteristics than the ones applied in this paper. Therefore, generalizations of the measurements are only justified for the studies where the considered traces can be represented by the traces we applied in this study.

Different sources and types of failures can result in different characterizations of the failure traces which are not covered in this study. Completeness of the employed set of traces is not the claim of this study. Our goal is to cover a large spectrum of input traces in order to emphasize that these characteristics have an impact on the performance of SHS.

The employed self-healing approaches are limited, and as a result, generalization of the findings cannot be a claim of this study. However, the selected self-healing approaches are shown in our previous works [43,58] to fairly represent the space of self-healing approaches as they spread between the two edge cases: the static approach is a non-optimal cost-effective approach with minimal runtime overhead, while the solver approach typically achieves optimal rewards but introduces large runtime overhead. Considering these edge cases, the u-driven approach is the case where these two edge cases meet and is shown to be both optimal and cost-effective. Another threat could be a selection bias to the choice of the self-healing approaches, as we only considered rule-based approaches. However, the decision enables comparability between the employed approaches. Overall, as there is no claim regarding the specific approach employed or the coverage of the complete spectrum of self-healing approaches, our findings are not affected by these limitations.

Construct Validity

The major threats to construct validity are the correctness of the simulation environment, our implementation of the self-healing approaches, and our choice of the failure traces. To address these threats, we used mRUBiS as our simulation environment, which has been accepted as an exemplar by the research community on self-adaptive software and has been extensively tested by students in the scope of four courses on self-adaptive software. The soundness of the employed self-healing approaches has been confirmed in our previous works [43,54,58]. Since the employed utility functions are the same for all the considered self-healing approaches, the specific way of constructing these utility functions is not a threat to the validity of the experiments, as none of the claims depend on the specific utility function or the absolute values of the measured reward. Therefore, reward is a proper measurement for the system performance.

6. Discussion

The conducted systematic literature review followed by an analytical assessments in this paper revealed that the evaluation of the SHS is treated naively by the current state-of-the-art. In order to obtain robust, conclusive, and reliable results from evaluation of a SHS, multiple reproducible experiment runs should be conducted.

Employing one failure trace as input for an under evaluation SHS (revealed by our SLR to be the common practice among the investigated studies) only supports a single experiment run and does not justify any claim on the robustness of the results. An individual failure trace only captures one possible future for the simulated SHS and fails to cover a large and representative spectrum of the input space. Employing such a trace results in inconclusive output for SHS under evaluation and lacks generality. Studies on evaluation of SHS can have generic credibility claims only if the results are tested for robustness in the presence of a large spectrum of the input space during multiple reproducible experiments. Therefore, multiple failure traces with volatile characteristics are required to support the required number of the experiments and improve the evidence.

Besides the outcome of the analytical assessment, the empirical assessment of Hypothesis 1 indicates that, even for probabilistic traces, wrong assumptions regarding the characteristics of the input trace can result in large prediction errors for performance of the self-healing approaches. Therefore, absolute performance predictions seem possible only if recorded real failure traces of enormous length or probabilistic traces fitted to real data are available. However, for less demanding design-time decisions such as the selection of alternative self-healing approaches, the empirical assessment of Hypothesis 2 indicates that having wrong assumptions regarding the characteristics of the failure occurrences can result in disadvantageous design-time decisions. Furthermore, for deployment-time decisions concerning optimization parameter tuning, the empirical assessment of Hypothesis 3 indicates that even if we only employ the evaluation for the decisions concerning the parameter tuning, employing the wrong characteristics for the occurrence of failures can result in disadvantageous deployment-time decisions. Finally, the empirical assessment of Hypothesis 4 indicates that employing multiple alternative input traces with volatile characteristics of the occurrence of failures can reduce the risk of premature disadvantageous design-time decisions.

Required Improvements

Conducting multiple reproducible experimental runs under controlled circumstances is required to obtain robust, conclusive, and reliable results from evaluation of a SHS. Generic credibility claims of the studies on evaluation of SHS are only justified if the results are tested for robustness in the presence of large spectrum of the input space. Therefore, multiple failure traces with volatile characteristics are required to support the claim and improve the evidence.

Taking into account that: (I) the characteristics of the SHS environment are always only known to a limited extend due to the usually rare nature of failures, (II) the characteristics of the SHS environment are also subject to change over time, and (III) we are interested in a robust solution that also performs well for all kinds of situations, it seems necessary to consider multiple alternative probabilistic traces when using evaluation to guide design-time and deployment-time decisions.

Moreover, the development of more competent self-healing approaches can be beneficial in mitigating the identified limitations. The findings of the experiments validating Hypothesis 2, Hypothesis 3, and Hypothesis 4 indicate that developing approaches that perform reasonably well in the presence of a large range of input traces (e.g., u-driven approach, see Table 4) which also do not present large sensitivity to the parameter tuning, or do not need tuning parameters, can help to reduce the observed problems.

However, this solution is not trivial, as it requires developing self-healing approaches that perform best under various volatile circumstances, and their performances cannot be further improved by tuning.

Secondly, inspired by the observations concerning the validation of Hypothesis 2 and Hypothesis 4, combining multiple self-healing approaches via a higher-order self-adaptive logic that selects the best performing approach for a given situation (e.g., select at runtime the best performing approach for a given FGS) can address the problem. Analogously, the observations concerning Hypothesis 3 indicate that employing a higher-order self-adaptive logic that selects the best performing parameter value for a given situation (e.g., tuning the parameter at runtime for given FGS) can help. Overall, similar to the rationale behind developments towards self-adaptive systems, delaying an activity—classically done offline either at design-time or deployment-time—to runtime may be beneficial, as at runtime more accurate information is available.

7. Conclusions and Future Work

Our SLR on the state-of-the-art for evaluating SHS revealed that while simulation-based experiments are pursued in 97% of the studies, only a few (two papers) use multiple input traces with volatile characteristics in their simulated evaluations. Our findings suggest that proper design and evaluation of SHS still remains an open issue, since critical elements for the design space of a SHS under evaluation are often missing.

We demonstrate the impact of the identified limitation concerning the input for SHS under evaluation through validating four hypotheses proposed in this paper. The hypotheses state that wrong assumptions regarding the characteristics of the input trace can result in: (I) large prediction errors for performance of self-healing approaches, and (II) disadvantageous design-time and (III) deployment-time decisions. They further propose that (IV) employing multiple alternative input traces with volatile characteristics of the occurrence of failures can reduce the risk of premature disadvantageous design-time decisions.

Our empirical assessments of the proposed hypotheses show that inaccurate assumptions regarding the characteristics of the input failure trace of a SHS at design-time can result in up to 138% performance prediction error of the self-healing approaches, and up to 51% performance loss, as they might result in disadvantageous and premature choices of the self-healing approaches. The experiments further demonstrate that wrong assumptions regarding the characteristics of the input trace at deployment-time can cause up to 575% performance loss. Finally, the empirical assessment of the hypotheses suggest that employing multiple input traces to steer the choice of the self-healing approaches at design-time reduces the risk of premature wrong decisions by 48%.

In the face of our findings, we recommend improving efforts such that multiple reliable and representative probabilistic failure traces supporting robustness testing of the measurements are employed to both steer the design-time and deployment-time decisions and support credible and robust measurements during SHS evaluation.

In future work, the authors plan to investigate mitigating the problem brought to light by this paper through employing an additional layer of self-awareness to a SHS which allows postponing certain design-time and deployment-time decisions to runtime, and therefore reduces the likelihood of performance loss due to inaccurate consideration of the potential input trace characteristics.

Author Contributions: Conceptualization, S.G. and H.G.; methodology, S.G. and H.G.; software, S.G.; validation, S.G.; writing–original draft preparation, S.G. and H.G.; writing–review and editing, S.G. and H.G.; visualization, S.G.; supervision, H.G.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SAS self-adaptive system
SHS self-healing system

References

1. Schmerl, B.; Andersson, J.; Vogel, T.; Cohen, M.B.; Rubira, C.M.F.; Brun, Y.; Gorla, A.; Zambonelli, F.; Baresi, L. Challenges in Composing and Decomposing Assurances for Self-Adaptive Systems. In *SEfSAS III: Assurances*; de Lemos, R., Garlan, D., Ghezzi, C., Giese, H., Eds.; Springer: New York, NY, USA, 2017.
2. Farahani, A.; Nazemi, E.; Cabri, G.; Rafizadeh, A. An evaluation method for Self-Adaptive systems. In Proceedings of the SMC '16 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Budapest, Hungary, 9–12 October 2016; pp. 002814–002820. [CrossRef]
3. Raibulet, C.; Fontana, F.A.; Capilla, R.; Carrillo, C. *Chapter 13—An Overview on Quality Evaluation of Self-Adaptive Systems*; Morgan Kaufmann: San Francisco, CA, USA, 2017; pp. 325–352.
4. Psaier, H.; Dustdar, S. A survey on self-healing systems: Approaches and systems. *Computing* **2011**, *91*, 43–73. [CrossRef]
5. de Sousa, A.O.; Bezerra, C.I.M.; Andrade, R.M.C.; Filho, J.M.S.M. Quality Evaluation of Self-Adaptive Systems: Challenges and Opportunities. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering, Salvador, Brazil, 2019*; ACM: New York, NY, USA, 2019; pp. 213–218.
6. Weyns, D.; Iftikhar, M.U.; Malek, S.; Andersson, J. Claims and Supporting Evidence for Self-adaptive Systems: A Literature Study. In Proceedings of the SEAMS '12 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Zurich, Switzerland, 4–5 June 2012; pp. 89–98.
7. Neti, S.; Muller, H.A. Quality Criteria and an Analysis Framework for Self-Healing Systems. In Proceedings of the 2007 International Workshop on Software Engineering for Adaptive and Self-Managing Systems, Minneapolis, MN, USA, 20–26 May 2007; IEEE Computer Society: Washington, DC, USA, 2007; p. 6. [CrossRef]
8. Pearson, S.; Campos, J.; Just, R.; Fraser, G.; Abreu, R.; Ernst, M.D.; Pang, D.; Keller, B. Evaluating and Improving Fault Localization. In Proceedings of the ICSE '17 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, 20–28 May 2017; pp. 609–620. [CrossRef]
9. Ghahremani, S.; Giese, H. Performance Evaluation for Self-Healing Systems: Current Practice & Open Issues. In Proceedings of the 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W), Umeå, Sweden, 16–20 June 2019; IEEE Computer Society: Los Alamitos, CA, USA, 2019; pp. 116–119.
10. Kaddoum, E.; Raibulet, C.; Georgé, J.P.; Picard, G.; Gleizes, M.P. Criteria for the Evaluation of Self-* Systems. In Proceedings of the SEAMS '10 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, Cape Town, South Africa, 1–8 May 2010; pp. 29–38. [CrossRef]
11. Gallet, M.; Yigitbasi, N.; Javadi, B.; Kondo, D.; Iosup, A.; Epema, D. A Model for Space-Correlated Failures in Large-Scale Distributed Systems. In *Euro-Par 2010—Parallel Processing: 16th International Euro-Par Conference, Ischia, Italy, 31 August–3 September 2010*; Proceedings, Part I; Springer: New York, NY, USA, 2010; pp. 88–100.
12. Vogel, T. mRUBiS: An Exemplar for Model-Based Architectural Self-Healing and Self-Optimization. In Proceedings of the SEAMS '18 International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Gothenburg, Sweden, 28–29 May 2018; ACM: New York, NY, USA, 2018. [CrossRef]
13. Patikirikorala, T.; Colman, A.; Han, J.; Wang, L. A Systematic Survey on the Design of Self-adaptive Software Systems Using Control Engineering Approaches. In Proceedings of the SEAMS '12 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Zürich, Switzerland, 4–5 June 2012; pp. 33–42.
14. Avizienis, A.; Laprie, J.C.; Randell, B.; Landwehr, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Secur. Comput.* **2004**, *1*, 11–33. [CrossRef]

15. Egyed, A. Architecture differencing for self management. In Proceedings of the 1st ACM SIGSOFT Workshop on Self-Managed Systems, Newport Beach, CA, USA, 31 October–1 November 2004; ACM: New York, NY, USA, 2004; pp. 44–48.
16. Koopman, P. Elements of the Self-Healing System Problem Space. In Proceedings of the WADS 2003, Waterloo, ON, Canada, 15–17 August 2013; pp. 31–36.
17. Heinis, T.; Pautasso, C.; Alonso, G. Design and Evaluation of an Autonomic Workflow Engine. In Proceedings of the Second International Conference on Autonomic Computing (ICAC'05), Seattle, WA, USA, 13–16 June 2005; pp. 27–38. [CrossRef]
18. Reinecke, P.; Wolter, K.; van Moorsel, A. Evaluating the adaptivity of computing systems. *Perform. Eval.* **2010**, *67*, 676–693. [CrossRef]
19. Cámara, J.; de Lemos, R.; Laranjeiro, N.; Ventura, R.; Vieira, M. Robustness-Driven Resilience Evaluation of Self-Adaptive Software Systems. *IEEE Trans. Dependable Secur. Comput.* **2017**, *14*, 50–64. [CrossRef]
20. Kitchenham, B.A.; Charters, S. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*; Technical Report EBSE-2007-01; Keele University: Staffordshire, UK, 2007.
21. Riganelli, O.; Micucci, D.; Mariani, L. Policy Enforcement with Proactive Libraries. In Proceedings of the SEAMS '17 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), Buenos Aires, Argentina, 22–23 May 2017; pp. 182–192. [CrossRef]
22. Schmitt, J.; Roth, M.; Kiefhaber, R.; Kluge, F.; Ungerer, T. Realizing Self-x Properties by an Automated Planner. In Proceedings of the ICAC '11 8th ACM international conference on Autonomic Computing, New York, NY, USA, 17–19 May 2011; pp. 185–186.
23. Ehlers, J.; van Hoorn, A.; Waller, J.; Hasselbring, W. Self-adaptive Software System Monitoring for Performance Anomaly Localization. In Proceedings of the ICAC '11 8th ACM international conference on Autonomic Computing, New York, NY, USA, 17–19 May 2011; pp. 197–200.
24. Salehie, M.; Tahvildari, L. A Coordination Mechanism for Self-healing and Self-optimizing Disciplines. In Proceedings of the SEAMS '06 2006 International Workshop on Self-Adaptation and Self-Managing Systems, Shanghai, China, 21–22 May 2006.
25. Camara, J.; de Lemos, R. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In Proceedings of the SEAMS '12 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Zürich, Switzerland, 4–5 June 2012; pp. 53–62.
26. Haupt, T. Towards Mediation-based Self-healing of Data-driven Business Processes. In Proceedings of the SEAMS '12 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Zürich, Switzerland, 4–5 June 2012; pp. 139–144.
27. Brown, A.B.; Redlin, C. Measuring the Effectiveness of Self-Healing Autonomic Systems. In Proceedings of the Second International Conference on Autonomic Computing (ICAC'05), Seattle, WA, USA, 13–16 June 2005; pp. 328–329. [CrossRef]
28. Griffith, R.; Kaiser, G. A Runtime Adaptation Framework for Native C and Bytecode Applications. In Proceedings of the ICAC '06 IEEE International Conference on Autonomic Computing, Dublin, Ireland, 2–6 June 2016; pp. 93–104. [CrossRef]
29. Carzaniga, A.; Gorla, A.; Pezz'e, M. Self-healing by Means of Automatic Workarounds. In Proceedings of the SEAMS '08 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems, Gothenburg, Sweden, 28–29 May 2018; ACM: New York, NY, USA, 2018; pp. 17–24.
30. Griffith, R.; Kaiser, G.; L'opez, J.A. Multi-perspective Evaluation of Self-healing Systems Using Simple Probabilistic Models. In Proceedings of the ICAC '09 6th International Conference on Autonomic Computing, Barcelona, Spain, 15 June 2009; pp. 59–60.
31. Casanova, P.; Garlan, D.; Schmerl, B.; Abreu, R. Diagnosing Architectural Run-time Failures. In Proceedings of the SEAMS '13 2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, San Francisco, CA, USA, 20–21 May 2013; IEEE Press: New York, NY, USA, 2013; pp. 103–112.

32. Angelopoulos, K.; Souza, V.E.S.; Mylopoulos, J. Dealing with Multiple Failures in Zanshin: A Control-theoretic Approach. In Proceedings of the SEAMS 2014 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Hyderabad, India, 3–4 June 2014; pp. 165–174.
33. Magalhaes, J.P.; Silva, L.M. SHOWA: A Self-Healing Framework for Web-Based Applications. *Trans. Auton. Adapt. Syst.* **2015**, *10*, 4:1–4:28. [CrossRef]
34. Piel, E.; Gonzalez-Sanchez, A.; Gross, H.; Van Gemund, A.J. Spectrum-Based Health Monitoring for Self-Adaptive Systems. In Proceedings of the SASO '11 5th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, Ann Arbor, MI, USA, 3–7 October 2011; pp. 99–108.
35. Di Marco, A.; Inverardi, P.; Spalazzese, R. Synthesizing Self-adaptive Connectors Meeting Functional and Performance Concerns. In Proceedings of the SEAMS '13 2013 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, San Francisco, CA, USA, 20–21 May 2013; IEEE Press: New York, NY, USA, 2013; pp. 133–142.
36. Albassam, E.; Porter, J.; Gomaa, H.; Menasce', D.A. DARE: A Distributed Adaptation and Failure Recovery Framework for Software Systems. In Proceedings of the ICAC '17 2017 IEEE International Conference on Autonomic Computing (ICAC), Columbus, OH, USA, 17–21 July 2017; pp. 203–208. [CrossRef]
37. Kantert, J.; Spiegelberg, H.; Tomforde, S.; Hähner, J.; Müller-Schloer, C. Distributed Rendering in an Open Self-Organised Trusted Desktop Grid. In Proceedings of the ICAC '15 2015 IEEE International Conference on Autonomic Computing, Grenoble, France, 7–10 July 2015; pp. 267–272. [CrossRef]
38. Kwak, K.J.; Baryshnikov, Y.M.; Coffman, E.G. Self-Organizing Sleep-Wake Sensor Systems. In Proceedings of the SASO '08 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, Venezia, Italy, 20–24 October 2008; pp. 393–402. [CrossRef]
39. Ishikawa, H.; Courbot, A.; Nakajima, T. A Framework for Self-Healing Device Drivers. In Proceedings of the SASO '08 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, Venezia, Italy, 20–24 October 2008; pp. 277–286. [CrossRef]
40. Baresi, L.; Guinea, S.; Saeedi, P. Self-managing Overlays for Infrastructure-less Networks. In Proceedings of the SASO '13 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems, Philadelphia, PA, USA, 9–13 September 2013; pp. 81–90. [CrossRef]
41. Stojnic, N.; Schuldt, H. OSIRIS-SR: A Safety Ring for self-healing distributed composite service execution. In Proceedings of the SEAMS '12 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Zürich, Switzerland, 4–5 June 2012; pp. 21–26. [CrossRef]
42. Pournaras, E.; Ballandies, M.; Acharya, D.; Thapa, M.; Brandt, B. Prototyping Self-Managed Interdependent Networks—Self-Healing Synergies against Cascading Failures. In Proceedings of the SEAMS '18 International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Gothenburg, Sweden, 28–29 May 2018; pp. 119–129.
43. Ghahremani, S.; Giese, H.; Vogel, T. Efficient Utility-Driven Self-Healing Employing Adaptation Rules for Large Dynamic Architectures. In Proceedings of the 2017 IEEE International Conference on Autonomic Computing (ICAC), Columbus, OH, USA, 17–21 July 2017.
44. Chan, K.S.M.; Bishop, J. The design of a self-healing composition cycle for Web services. In Proceedings of the SEAMS '09 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, Atlanta, GA, USA, 31 August 2009; pp. 20–27.
45. Duan, S.; Babu, S. Guided Problem Diagnosis through Active Learning. In Proceedings of the ICAC '08 2008 International Conference on Autonomic Computing, Chicago, IL, USA, 2–6 June 2008; pp. 45–54. [CrossRef]
46. Bohra, A.; Neamtiu, I.; Gallard, P.; Sultan, F.; Iftode, L. Remote repair of operating system state using Backdoors. In Proceedings of the ICAC '04 International Conference on Autonomic Computing, New York, NY, USA, 17–18 May 2004; pp. 256–263. [CrossRef]
47. Klopper, B.; Honiden, S.; Meyer, J.; Tichy, M. Planning with Utility and State Trajectory Constraints in Self-Healing Automotive Systems. In Proceedings of the SASO '10 2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, Budapest, Hungary, 27 September–1 October 2010; pp. 74–83. [CrossRef]

48. Renz, W.; Preisler, T.; Sudeikat, J. Mesoscopic Stochastic Models for Validating Self-Organizing Multi-Agent Systems. In Proceedings of the SASO '12 2012 IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems Workshops, Lyon, France, 10–14 September 2012; pp. 119–126. [CrossRef]
49. Audrito, G.; Casadei, R.; Damiani, F.; Viroli, M. Compositional Blocks for Optimal Self-Healing Gradients. In Proceedings of the SASO '17 2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Tucson, AZ, USA, 18–22 September 2017; pp. 91–100. [CrossRef]
50. Ippoliti, D.; Zhou, X. A Self-tuning Self-optimizing Approach for Automated Network Anomaly Detection Systems. In Proceedings of the ICAC '12 9th International Conference on Autonomic Computing, San Jose, CA, USA, 16–20 September 2012; pp. 85–90.
51. Haesevoets, R.; Weyns, D.; Holvoet, T.; Joosen, W. A formal model for self-adaptive and self-healing organizations. In Proceedings of the SEAMS '09 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, Atlanta, GA, USA, 31 August 2009; pp. 116–125.
52. Anaya, I.D.P.; Simko, V.; Bourcier, J.; Plouzeau, N.; J'ez'equel, J.M. A Prediction-driven Adaptation Approach for Self-adaptive Sensor Networks. In Proceedings of the SEAMS 2014 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Hyderabad, India, 3–4 June 2014; pp. 145–154.
53. Jánicke, M.; Tomforde, S.; Sick, B. Towards Self-Improving Activity Recognition Systems Based on Probabilistic, Generative Models. In Proceedings of the ICAC '16 2016 IEEE International Conference on Autonomic Computing (ICAC), Wurzburg, Germany, 17–22 July 2016; pp. 285–291. [CrossRef]
54. Ghahremani, S.; Adriano, C.M.; Giese, H. Training Prediction Models for Rule-Based Self-Adaptive Systems. In Proceedings of the ICAC '18 15th IEEE International Conference on Autonomic Computing, At Trento, Italy, 3–7 September 2018; pp. 187–192. [CrossRef]
55. C'amara, J.; Garlan, D.; Schmerl, B.; Pandey, A. Optimal Planning for Architecture-based Self-adaptation via Model Checking of Stochastic Games. In Proceedings of the SAC '15 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, 13–17 April 2015; ACM: New York, NY, USA, 2015; pp. 428–435.
56. Kephart, J.O.; Walsh, W.E. An Artificial Intelligence Perspective on Autonomic Computing Policies. In Proceedings of the IEEE POLICY'04 Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, Yorktown Heights, NY, USA, 7–9 June 2004; pp. 3–12. [CrossRef]
57. Ghahremani, S.; Giese, H.; Vogel, T. Towards Linking Adaptation Rules to the Utility Function for Dynamic Architectures. In Proceedings of the IEEE SASO '16 2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems (SASO), Augsburg, Germany, 12–16 September 2016; pp. 142–143.
58. Ghahremani, S.; Giese, H.; Vogel, T. Improving Scalability and Reward of Utility-Driven Self-Healing for Large Dynamic Architectures. *ACM Trans. Auton. Adapt. Syst.* **2020**, *14*, 12:1–12:41. [CrossRef]
59. Seborg, D.E.; Mellichamp, D.A.; Edgar, T.F.; Doyle, F.J. *Process Dynamics and Control*, 3rd ed.; John Wiley & Sons: New York, NY, USA, 2011.
60. Wongpiromsarn, T.; Topcu, U.; Murray, R.M. Receding Horizon Temporal Logic Planning. *IEEE Trans. Autom. Control* **2012**, *57*, 2817–2830. [CrossRef]
61. Li, L.; Negenborn, R.R.; Schutter, B.D. Intermodal freight transport planning—A receding horizon control approach. *Transp. Res. Part C Emerg. Technol.* **2015**, *60*, 77–95. [CrossRef]
62. Castillo, X.; McConnel, S.R.; Siewiorek, D.P. Derivation and Calibration of a Transient Error Reliability Model. *IEEE Trans. Comput.* **1982**, *C-31*, 658–671. [CrossRef]
63. Tang, D.; Iyer, R.K. Dependability measurement and modeling of a multicomputer system. *IEEE Trans. Comput.* **1993**, *42*, 62–75. [CrossRef]
64. Iyer, R.K.; Butner, S.E.; McCluskey, E.J. A Statistical Failure/Load Relationship: Results of a Multicomputer Study. *IEEE Trans. Comput.* **1982**, *C-31*, 697–706. [CrossRef]
65. Heath, T.; Martin, R.P.; Nguyen, T.D. Improving Cluster Availability Using Workstation Validation. *SIGMETRICS Perform. Eval. Rev.* **2002**, *30*, 217–227. [CrossRef]

66. Zhang, Y.; Squillante, M.S.; Sivasubramaniam, A.; Sahoo, R.K. Performance Implications of Failures in Large-Scale Cluster Scheduling. In *Job Scheduling Strategies for Parallel Processing: 10th International Workshop, JSSPP 2004*; Revised Selected Papers; Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 233–252. [CrossRef]
67. Iosup, A.; Dumitrescu, C.; Epema, D.; Li, H.; Wolters, L. How Are Real Grids Used? The Analysis of Four Grid Traces and Its Implications. In Proceedings of the GRID '06 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain, 28–29 September 2006; IEEE Computer Society: Washington, DC, USA, 2006; pp. 262–269. [CrossRef]
68. Iosup, A.; Jan, M.; Sonmez, O.; Epema, D. On the Dynamic Resources Availability in Grids. In Proceedings of the 8th IEEE/ACM International Conference on Grid Computing, Austin, TX, USA, 19–21 September 2007.
69. Kondo, D.; Javadi, B.; Iosup, A.; Epema, D. The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In Proceedings of the CCGRID '10 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, Melbourne, Australia, 17–20 May 2010; pp. 398–407. [CrossRef]
70. Davison, A.C.; Kuonen, D. An Introduction to the bootstrap with applications in R. *Stat. Comput. Stat. Graph. Newsl.* **2002**, *13*, 6–11.
71. Efron, B.; Tibshirani, R.J. *An Introduction to the Bootstrap*; Chapman & Hall: New York, NY, USA, 1993.
72. Perino, N. A Framework for Self-healing Software Systems. In Proceedings of the ICSE '13 2013 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, 18–26 May 2013; pp. 1397–1400.
73. Sestoft, P. Microbenchmarks in Java and C#; Lecture Notes; 2013. Available online: https://www.itu.dk/people/sestoft/papers/benchmarking.pdf (accessed on 26 February 2020).