Testing Self-Adaptive Software with Probabilistic Guarantees on Performance Metrics

Claudio Mandrioli
claudio.mandrioli@control.lth.se
Lund Univeristy

Martina Maggio
maggio@cs.uni-saarland.de
Saarland University & Lund Univeristy

ABSTRACT

This paper discusses the problem of testing the performance of the adaptation layer in a self-adaptive system. The problem is notoriously hard, due to the high degree of uncertainty and variability inherent in an adaptive software application. In particular, providing any type of formal guarantee for this problem is extremely difficult. In this paper we propose the use of a rigorous probabilistic approach to overcome the mentioned difficulties and provide probabilistic guarantees on the software performance. We describe the set up needed for the application of a probabilistic approach. We then discuss the traditional tools from statistics that could be applied to analyse the results, highlighting their limitations and motivating why they are unsuitable for the given problem. We propose the use of a novel tool – *the scenario theory* – to overcome said limitations. We conclude the paper with a thorough empirical evaluation of the proposed approach, using two adaptive software applications: the Tele-Assistance Service and the Self-Adaptive Video Encoder. With the first, we empirically expose the trade-off between data collection and confidence in the testing campaign. With the second, we demonstrate how to compare different adaptation strategies.

CCS CONCEPTS

• **Software and its engineering → Empirical software validation**; **Software testing and debugging**; • **Social and professional topics →** *Software selection and adaptation.*

KEYWORDS

Testing, Self-Adaptive Software, Autonomous Systems

ACM Reference Format:
Claudio Mandrioli and Martina Maggio. 2020. Testing Self-Adaptive Software with Probabilistic Guarantees on Performance Metrics. In *ESEC/FSE 2020, The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 8 - 13 November, 2020, Sacramento, California, United States.* ACM, New York, NY, USA, 12 pages. https://doi.org/acmDOI--CHANGEME

1 INTRODUCTION

Software systems are affected by uncertainty that alters their behaviour and can render their performance unpredictable. Adaptation layers were introduced in software as a viable solution to deal with performance fluctuations and minimise the effect of uncontrolled changes [17, 18, 57]. This makes software self-adaptive. The idea behind self-adaptive software is to have a layer responsible for observing behavioural changes and taking counteractions. This can guarantee more stable and predictable software performance in terms of non-functional software behaviour [28, 48], e.g., lower response times, or higher reliability.

Adaptation can be implemented using different methodologies; some of them provide guarantees based on formal models [29, 34, 48], others are empirically proven effective [23, 49, 60]. In both cases there is a need for appropriate performance testing of the system composed of the software and its adaptation layer. The presence of an adaptation layer opens up the possibility that in the same exact condition the software will behave differently, depending on its past behaviour and accumulated knowledge. It is necessary to conduct empirical validation of satisfactory behaviour to verify the correctness of the system and adaptation-layer implementation [69]. In addition, it is important to quantify the achievable performance.

In general, testing is a crucial aspect of software development. For self-adaptive software, the testing process is complicated by the presence of the adaptation layer [7, 10, 33, 62]. Self-adaptive systems testing is intrinsically hard, due to the extreme variability and uncertainty involved in the software execution [5, 8, 20, 62]. In fact, the adaptation layer explicitly reacts to the uncertainty, and may influence it for the future. This creates a *loop* around the software [57]. In the context of uncertainty and adaptation, this paper's challenge is to achieve and maintain formal guarantees on non-functional aspects of the software execution, like reliability and response times.

Research Challenges: The adaptation layer and the presence of uncertainty impose specific challenges for testing. Triggered by the environmental variability, the adaptation generates changes in the system - and this changing nature makes it difficult (and in many cases impossible) to *exhaustively* guarantee its correct behaviour [27, 47, 50, 62, 65, 68]. The adaptation also creates a difficulty in the performance quantification and in determining the testing sufficiency and effectiveness [20, 62].

As an example, consider testing a web-application that can run on different servers with different and time-varying performance results. Not all of the servers can provide the same reliability. The adaptive layer should choose dynamically which server to use, in order to maximise the overall reliability. In general, it is not possible to guarantee that the application is always reliable, since any server may fail. Also, the actual reliability will depend significantly on the specific servers, and on their performance. As a consequence, when testing the system, any evaluation of its reliability is heavily affected by the specific test cases. Determining which tests are sufficient

ESEC/FSE 2020, 8 - 13 November, 2020, Sacramento, California, United States
© 2020 Association for Computing Machinery.
ACM ISBN acmISBN–CHANGE ME…$15.00
https://doi.org/acmDOI--CHANGEME

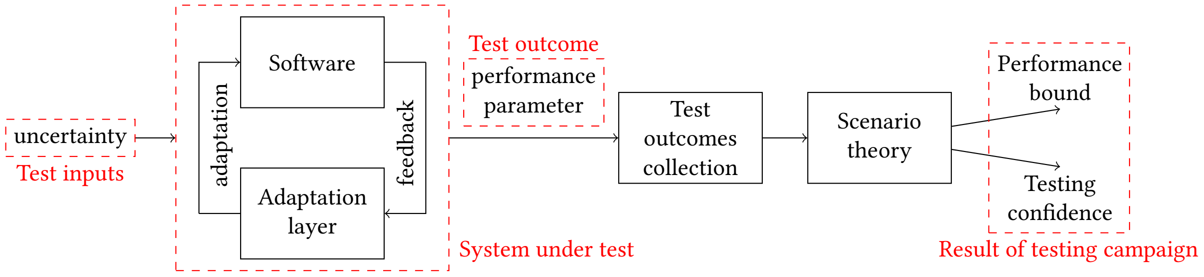

Figure 1: Overview of the proposed approach.

and when it is possible to stop the testing process becomes challenging. To be precise, this paper addresses the following research challenges [20, 62]:

- **CH1**: Definition of what type of guarantees can be given for self-adaptive software.
- **CH2**: Quantification of the mentioned guarantees.
- **CH3**: Quantification of the testing sufficiency and effectiveness (or testing adequacy).

Contribution: In this paper we address the three mentioned research challenges by leveraging a rigorous probabilistic approach [9, 22]. The probabilistic approach is beneficial in two ways: (i) it allows the efficient exploration of large input and configuration spaces [54], and (ii) it can provide a quantification of its own adequacy. In the field of probability theory, the testing adequacy is called *confidence*.

As discussed in the research challenges above, the uncertain nature of self-adaptive systems does not allow for the definition of strict guarantees. This limitation mainly arises from the large (and possibly infinite) number of combinations of inputs that can be provided to the system [20, 62]. Despite this, *we need to test self-adaptive systems when said variability is present, in order to trigger the adaptive behaviour.* Leveraging a probabilistic approach, the uncertainty and variability can efficiently be explored using randomised inputs, independent of their size [22, 54]. As a consequence, the measured performance metric must be treated as a random quantity, and requires statistical evaluation. We therefore enter the domain of probabilistic guarantees [1, 9].

In this work, we focus on the evaluation of probabilistic bounds for a given performance metric. Our aim is testing what is the value of this performance parameter that the adaptive software can guarantee in the majority of its execution environments. We formally define majority in a probabilistic fashion, e.g., that a given performance bound will hold in 99% of the execution instances. We also quantify the confidence that we can claim, i.e., the adequacy of our testing campaign. High confidence means a high probability that we performed a sufficient number of randomly generated tests to sustain our claim. In some sense, this is analogous to a *coverage criterion* – a reference for choosing when to stop the testing campaign.

We discuss traditional tools from statistics and highlight their limitations for testing self-adaptive software. We overcome these limitations using a tool called *scenario theory* [12]. The scenario theory was developed in the field of robust control but can actually be applied to a very general class of problems. In this paper we show how to apply it to the problem of testing self-adaptive software.

Experimental Evaluation: To support our claims, we use our methodology to test the behaviour of two self-adaptive software applications: the Tele-Assistance System [70], and the Self-Adaptive Video Encoder [45]. We show how our methodology can be used to: (i) rigorously quantify the adaptation performance, (ii) evaluate the trade-off between the number of performed tests and the confidence in the testing campaign, and (iii) compare adaptation strategies.

Paper Structure: In Section 2 we provide an overview of our proposed testing approach. Section 3 discusses related work. Section 4 presents our methodology and describes how it overcomes the limitations of classical statistical testing. Section 5 presents experimental results, and Section 6 describes the limitations of our proposal. Finally, Section 7 concludes the paper.

2 APPROACH OVERVIEW

In this Section we provide an overview of our testing approach (shown in Figure 1). In particular, we discuss: (i) the definition of *test inputs*, (ii) the definition of the *test outcome*, and (iii) the evaluation of the *results of the testing campaign.* In Section 4 we discuss in detail how to apply the scenario theory to evaluate the test outcomes and obtain the *performance bound* and *testing confidence.*

The objective of our testing campaign is to empirically provide guarantees on the system behaviour. These guarantees should be general and independent from the specific test cases. Practically, we want independence from the variability and uncertainty that affects the executed tests. We obtain this by performing different random tests, each of which represents a possible system realisation. We then statistically evaluate the results of the testing campaign.

Each of the test cases is defined by randomly picking a possible instance of the system (e.g., for the web-application in the introduction, by picking servers with some given reliability values). We randomly select the *test inputs* (Figure 1). For example, if our adaptive system has to recognise server failures in a web-application, we define test cases in which the servers fail at random points in time. If the performance evaluation is carried out rigorously, it provides a result independent from the specific failure times observed in the different test cases.

The randomly selected test inputs are fed into the system under test, as described by the arrow connecting the test inputs and the system under test in Figure 1. In order to evaluate the effectiveness of the adaptation strategy, we define a *performance parameter.* The performance parameter is a quantity that (i) can be measured from the execution of a test case, and (ii) is higher or lower, according to the degree at which the adaptation strategy has achieved its goals. In the web-application case mentioned above, this parameter could be for example the average time spent recovering from server failures over the whole test duration. The key intuition is that this performance parameter is itself a random variable [9], and we can therefore use tools from statistics to predicate on its value.

We collect the outcomes of the tests and evaluate them using the scenario theory. By leveraging this theory we obtain *probabilistic bounds* on the chosen performance metric and a *testing confidence*. The probabilistic bounds are in the form of a minimum performance that is guaranteed in a high percentage of the cases. The testing confidence is instead given as a probability. To be precise, the confidence is the probability that we have missed relevant test cases that would have changed the obtained bound. Continuing with the example above we would obtain a bound like "*the time it takes to recover from a server failure is on average less than* 42 *seconds in* 97% *of the cases, with a* 95% *confidence*". This means that we have a $100 - 95 = 5\%$ probability of having missed a relevant test case. If the confidence is not sufficient, the scenario theory allows to directly compute how many additional tests are needed to increase it to the desired level.

3 RELATED WORK

This section discusses how this work is connected to the existing research literature. To start, we present related work in the software testing research field. Then we present the traditional statistical tools used to extract probabilistic properties from test outcomes.

3.1 Testing of Adaptive Systems

Our work connects to different areas of the existing software testing literature: (i) testing of self-adaptive and context-aware systems, (ii) testing in the presence of environmental dependencies, (iii) fuzz testing, and (iv) testing for probabilistic guarantees.

The problem of testing an adaptive software – in some cases also called *context-aware* software [47, 66] – is not a new challenge for the software testing community [20, 62]. We split the work that addresses the testing of self adaptive software in *design-time* and *run-time* approaches. For self-adaptive software, the design-time approaches include SIT [51] and TestDAS [59]. SIT [51] proposes a test case generation technique for self-adaptive applications. The sampling of the input space is based on an interactive model of the application that is being tested. TestDAS [59] focuses on triggering the adaptations during the test cases. It leverages models of the software behaviour that are defined in advance by the programmer. Context-aware software is close to self-adaptive software, and there is a significant amount of work addressing the problem of testing context-aware applications [46, 47, 66, 72]. The self-adaptive (or context-aware) software observes the execution environment and selects actions to be performed based on the result of the observation phase. The research effort for context-aware software goes in the direction of generating test cases that trigger the context-aware software layer [46, 47, 72]. In [72], automatically generated bigraphs are used to model the interactions between the environment and the software, and to generate the test cases. In [46] the authors propose a framework for automatically generating test cases with high-level test data.

Our proposal is different from previous work on context-aware and self-adaptive software testing, since in our case the interaction with the environment only needs a probabilistic characterisation, and no further modelling effort. Moreover, in our contribution, the number of test cases does not depend on how the interaction with the environment is performed. This is important since it allows our method to scale with the amount of interaction between software and environment.

The literature on software testing also includes efforts to develop run-time testing methodologies for adaptive software [18, 38, 53]. Generally speaking, there is a need to develop models for verification and validation at run-time [18]. This need is caused by the ever-changing nature of the environment the adaptive software operates in. We describe our approach for design-time testing, but in principle[1] the resulting method can be applied during the run-time execution of the software application, since it only requires data collection and analysis. A clear difference between our work and the related literature is that we develop a probabilistic approach.

In our work, we use statistical tools to evaluate the performance of the adaptation layer of a self-adaptive software, independently from changes in the environment. Previous work also addressed the problem of testing a software regardless of its environmental dependencies [4, 35]. These works aim at decoupling the tests outcomes from such dependencies. To test the adaptation layer, we need to preserve the dependency on the environment, since it triggers the need for adaptation. However, we aim at obtaining an evaluation that is general with respect to the environment changes.

The approach we propose in this paper is based on random sampling of the system inputs and environment scenarios. This practice is known to the software testing community [3, 16], and is often called fuzz testing [9, 64, 71, 73]. The literature focuses on using random generation for achieving adequate exploration of the software behaviour, e.g., code coverage [64]. We take inspiration from fuzz testing, and use random sampling with two different objectives: (i) decoupling given inputs or environmental scenarios from performance parameters that indicate how well the adaptation layer is performing, and (ii) obtaining a probabilistic characterisation of the performance metric.

Probabilistic guarantees have been explored [36, 37, 56]. In some cases this exploration targeted approximate computing [5, 21, 22, 41], which is not the subject of this study. Some existing work target service-oriented software architectures [13] and how to combine the probabilistic guarantees given by the different services to obtain guarantees for the complete system [37, 56]. However, no prior work targets dynamic behaviour (i.e., behaviour that changes during the execution of the software, as it is the case with the adaptation layer) and adaptive software, which is the focus of our work.

3.2 Tools from Statistics

In this section, we recall traditional tools from statistics, that could be used to analyse the result of tests and provide statistical guarantees on the software behaviour. In particular, we discuss the limitations of existing methods that we aim to overcome.

[1]The requirement to apply our approach at run-time is that the run-time tests are considered random independent tests. Testing the system continuously might not guarantee independence. This can be solved (for example) by introducing a delay between consecutive tests.

Monte Carlo Sampling (MC): Monte Carlo (MC) methods [54] use repeated random sampling and simulation to numerically predict the value of parameters. The parameters are unknown, and usually no exact analysis can be carried out (for example because there are too many random variables, i.e., too much uncertainty). Nowadays, MC methods are employed in many different fields, from optimisation [55] to decision making [39]. MC methods leverage the Central Limit Theorem [40] as a main mathematical result. The theorem discusses the mean of a random variable with an arbitrary probability distribution, under the assumption that the variance of the distribution is finite. The theorem states that, if one draws infinitely many samples from the random variable, the distribution of the arithmetic mean of the samples asymptotically converges to a normal distribution, regardless of the original variable distribution.

The application of MC approaches allows to conduct an arbitrary number n of tests and measure the random variable X, obtaining a set of outcomes $\{x_1, \dots, x_n\}$. Then it is possible to determine the mean value $\bar{x}$ as the arithmetic average of the tests outputs,

$$\bar{x} = {}^1/_n \sum_{i=1}^{n} x_i. \tag{1}$$

The computed arithmetic mean $\bar{x}$ is also a random variable. The Central Limit Theorem guarantees that its distribution converges to a normal distribution for increasing n, i.e., $\bar{x} \sim \mathcal{N}(E[X], {}^{\sigma^2}/_n)$, where $E[X]$ is the expected value of the random variable X and σ^2 is the variance of X. When n is big enough, the observed mean value converges to the actual expected value for the quantity of interest. This result is well-known in statistics and it holds irrespective of the specific software application under test. In fact, convergence is guaranteed independently from the probability distribution of the performance metric. However, there is no general result on the speed of the convergence and it is therefore application-dependant. With MC sampling, the significance of the test results and the choice of n is therefore left as an arbitrary choice to the testing engineer. The confidence in the final result is dependant on the variance σ^2 defined above. This quantity is unknown and has to be estimated, adding one degree of uncertainty to the testing process.

MC methods have found limited use in the context of software testing [42, 61]. None of these works focuses on the testing of self-adaptive software. In [61] MC methods are used to test the reliability of a software system, while [42] generally discusses how MC methods can be applied to software testing.

Extreme Value Theory (EVT): The Extreme Value Theory [19] (EVT) studies a random variable around the tails of its distribution. It could therefore be used when we specifically want to analyse the software's *worst-case* behaviour, e.g., what is the maximum memory occupation of a program. The theory is nowadays widely adopted to study rare phenomena such as earthquakes, quantitative risks in finance, but also extreme events in engineering [15, 58].

The role of the Central Limit Theorem for MC sampling is taken by the Fisher–Tippett–Gnedenko Theorem [30] for the EVT. The Fisher–Tippett–Gnedenko theorem defines the family of distributions to which the maximum value of a set of samples converges. The family of distributions is called the Generalised Extreme Value Distribution [19]. To apply EVT, we can look at a set of data (in our case the performance parameters obtained from the test cases) and extract a set of samples that belong to the tail of the dataset – i.e., a set of *maxima*. We then fit the the Generalised Extreme Value Distribution to the extracted maxima. In this way, we can obtain a probability distribution for the extreme value of the performance metric that could be observed in future executions of the system.

EVT presents similar limitations compared to MC approaches [25]. EVT uses an arbitrary number of samples from the distribution of interest. Moreover, the choice of which and how many samples can be considered as the maxima is also arbitrary. There are also no results on the rate of convergence of the samples to the Generalised Extreme Value Distribution. In general, this convergence is known to be slow since it requires several observations of events belonging to the tail of the distribution, and therefore intrinsically rare. Finally, as MC, EVT requires finite variance of the parameter that is sampled.

4 METHODOLOGY

In this section, we describe our approach to obtain probabilistic guarantees and its theoretical underpinning.

4.1 Limitations of Traditional Statistics

In Section 3.2 we described the traditional tools from statistics that could be used to obtain probabilistic guarantees when testing self-adaptive software: MC and EVT. Both those methodologies suffer from limitations that make them inconvenient for analysing the results of the testing campaign – i.e. being used in place of the "Scenario theory" block in Figure 1. These limitations are: (i) arbitrary choice of testing parameters, (ii) unknown, case-dependent, testing confidence (or *testing adequacy*), and (iii) assumption that the variance of the measured quantity is finite.

MC and EVT use an arbitrary number of samples for the desired estimation. The MC approach assumes that the set of samples is large enough that the Central Limit Theorem holds [54], and the EVT similarly relies on the convergence of the maxima samples to the Extreme Value Distribution [26]. Unfortunately, in both the theories, there is no general way to define how many samples are needed to achieve convergence.

The impossibility of quantifying the convergence to the Gaussian and Extreme Value Distributions has another relevant implication for the testing problem. If the desired testing confidence is not reached, it is impossible to quantify how many extra tests are needed to reach it. In other words, we cannot know *a priori* how the confidence will change when performing one extra test.

Another assumption needed by both EVT and MC sampling is that the performance parameter has finite variance. In practice, this means that either the probability of it being infinite must be very low, or that the parameter can only take finite values. Suppose we are trying to assess the worst-case execution time of a software function. The presence of a bug could cause the processor to stall and the function to never terminate. As long as the occurrence of this bug is sporadic, it is possible to use EVT and MC to determine metrics on the execution time. However, if the bug is triggered more often, the (higher) probability of an infinite execution time would prevent us from applying EVT and MC methods.

Our proposal overcomes these limitations by formulating the testing problem as an infinite optimisation problem and solving it using the *scenario theory*.

4.2 Scenario Theory for Software Testing

The *scenario approach* [12] was developed in the field of robust control [31]. However, it is more generally applicable than control design. It provides a method to solve *infinite convex optimisation problems.* Infinite convex optimisation problems are a class of optimisation problems that appear frequently in robust control design. However, they are also classically found in other fields, like decision making, finance, and management [11, 52]. The contribution of this paper is the formulation of the testing problem with the scenario approach and the study of the results that can be obtained for self-adaptive software. We will show that this allows us to overcome the research challenges presented in Section 1 .

In our testing problem, we want to find bounds for a performance parameter of an adaptive system (i.e., of the software and a given adaptation strategy implemented on top of it). In general, finding a safe and very pessimistic bound on what the software can achieve is trivial. The interesting question is how much we can move this bound toward higher performance. This problem can be formulated as: we would like to maximise the value of the performance parameter that we can safely guarantee when using a given adaptation algorithm.

The evaluation of this performance bound can therefore be seen as an optimisation problem. Solving optimisation problems means finding the extreme value of a quantity, either the highest or the lowest possible. In the following sections we introduce optimisation problems, the scenario theory, and how they can be used to bound the performance of a self-adaptive software.

Optimisation Problems: Optimisation problems are defined by: (i) one or more decision variables, (ii) a cost function, and (iii) a set of constraints. The decision variables are the quantities we can choose and change. The cost function is the quantity we would like to maximise or minimise, and it should be a function of the decision variables. The constraints are statements about the decision variables that we want our final solution to satisfy. An example of a problem that can be formulated as an optimisation problem is the travelling salesman problem [2]. A salesman needs to determine a route to visit a given number of cities, minimising the travelling distance. The decision variables are the segments to add to the path (from one city to the next), the cost function is the total travelled distance, and the constraint is that all the cities in the given list are visited at least once.

In our proposed testing methodology the decision variable is the worst-case performance of the adaptation strategy (i.e. the best value of the performance metric that we can safely guarantee), the cost function is the worst-case performance itself, and each of the test outcomes is a constraint. The performance bound evaluation therefore becomes the following optimisation problem: *maximise the performance that can always be guaranteed, under the constraint that it cannot exceed what is experienced in the conducted tests.*

Being even more practical and using the web application example from Section 1, suppose we want to provide guarantees on its maximum response time thanks to the adaptation strategy. We conduct a certain number n of tests. Each test is composed of servicing 1000 requests in random execution instances of the overall system, and monitoring their response times. We record the average response times in the vector $\mathbf{r} = \{r_1, r_2, \dots, r_n\}$. Where r_i is the average response time of the web application for the 1000 requests of the $i-th$ test. These values are constraints on what the software can achieve. We then take the maximum element of the vector as our worst-case performance metric, $w_{\max} = \max\{r_1, r_2, \dots, r_n\}$. If we tested all the possible execution cases, we could then say that we guarantee that the response time will be lower than the maximum value $w_{\max}$. However, for self-adaptive software the set of possible execution cases is likely infinite.

Ideally, if we could perform an infinite number of tests, we would test the system in every possible situation. In this way, we could obtain an exact evaluation of the worst case behaviour of the system. In practice, this is apparently not achievable, and we have to rely on only a finite number of tests. Despite this, when the number of tests is sufficiently large, it will still provide significant information about the general case.

Infinite Optimisation Problems: If we cast our (ideal) testing problem into an optimisation problem, we would have infinite constraints (the infinite test cases). For our web application example this would mean performing an infinite number of tests and obtaining the real bound.[2] Unfortunately, solving an optimisation problem with an infinite set of constraints is not always possible (or desirable). Similarly, in our testing problem, we cannot perform infinite tests.

Scenario theory: The scenario theory [12] addresses the problem of solving an infinite optimisation problem while accounting only for a finite number of the constraints. The theory provides probabilistic guarantees on the generality of the solution. The scenario approach is used to solve infinite optimisation problems. The approach is to transform the infinite-sized problem into a finite-sized problem by *randomly* sample a finite number of constraints from the infinite set of possible ones. Then, it is possible to solve the optimisation problem accounting only for the finite set of sampled constraints. The scenario theory allows then to quantify the uncertainty and the guarantees that are lost by only considering the finite set.

In the testing of our web application, this corresponds to obtaining the probabilistic guarantee that the average response time is lower or equal to $w_{\max}$ in a high percentage of the cases. This means that, with high probability, the future executions of the web application would not result in a higher average response time, i.e., $\mathrm{P}(r_m \leq w_{\max} \mid m > n) = p_w \approx 1$.

Using the scenario theory, we can compute the probability p_w that the solution – computed using the finite set – does not satisfy all the constraints in the possibly infinite set. For our testing problem, this means that we evaluate the worst-case performance using only a finite number n of test results. We then compute the probability that the obtained worst-case value $w_{\max}$ holds for all of the infinite tests that we could possibly run – i.e., we compute the probability

[2]Solving an optimisation problem with an infinite number of constraints can also be seen as robustly solving an uncertain optimisation problem. In an uncertain optimisation problem, the constraints belong to a given set (in our case infinite) but it is not known which of them we have to account for. Solving the problem robustly means finding a solution that accounts for all the possible constraints in the infinite set. This ensures that the solution satisfies all the constraints that are relevant for the problem, even though they are only a subset of the infinite set. Analogously, finding a bound on the performance of an uncertain system (like adaptive ones) means looking for a performance that we can always guarantee despite the different possible realisations of the system.

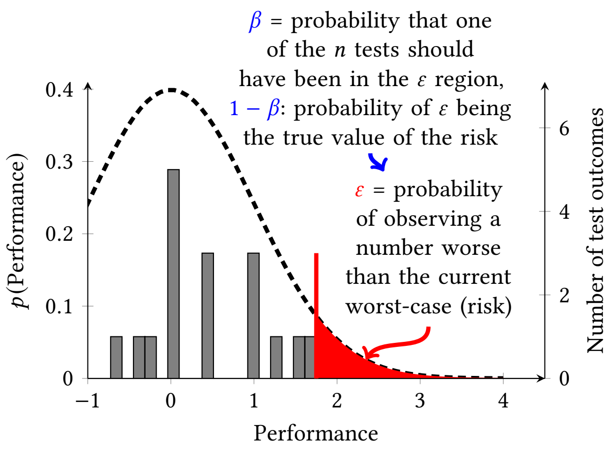

Figure 2: Graphical representation of the scenario parameters ε and β. In this case poor performance of the system is captured by high values of the performance parameter while low values correspond to more desirable performance.

that in future tests we would obtain a worse value than $w_{\max}$, which is obtained using the first n tests, i.e., that $\exists m > n \mid w_m > w_{\max}$.

In our specific optimisation problem for testing, we only have one decision variable (the evaluation of our worst-case). We now state the relevant result of the scenario theory in that case.[3] We denote with ε the probability of *observing (in future executions) a performance value that is worse than the observed worst-case up to n tests* (i.e., $\varepsilon = 1 - p_w$). In the original optimisation framework this is the probability of not satisfying all the infinite constraints.

Using the scenario theory, we can evaluate the probability that, in our n test cases, we could have *missed a test case with a worse performance than the obtained bound*. We call this probability β and it is computed from ε and n as

$$(1-\varepsilon)^n = \beta. \tag{2}$$

In the original optimisation problem, ε quantifies the probability that a new (randomly picked) constraint taken from the infinite set would invalidate the solution found using the finite set. In our testing analogy, ε is a quantification of how *tight* we want our bound to be. Choosing a lower probability ε means having a tighter bound, and choosing a higher value means that we allow for higher risk of not having observed the *true* worst-case. We remark that we can arbitrarily choose ε, but this will result in different degrees of confidence β that we can have in the obtained result.

The probability β can be seen as a quantification of how confident we are of our testing campaign result. A lower value of β implies that we are more confident and a higher value represents a higher probability that the final result is not correct. A tighter worst-case bound (lower ε), in fact, results in a higher β, a higher probability that we could have "missed" a relevant test case (constraint) in our sampling. In this sense, β can be seen as a *coverage* parameter, since it quantifies our confidence of having explored enough of the possible instances of the self-adaptive software behaviour.

[3]We omit the complete formula for an arbitrary number of decision variables, since it is not of interest in our case.

Figure 2 shows a graphical interpretation of the probabilities ε and β. The dashed line shows the true and unknown probability distribution of the performance parameter. The histogram represents the observations that we obtained when measuring the performance of the system in our tests (i.e., our test results). The purple bar indicates the worst case obtained during the testing campaign. The purple area has size ε, i.e., ε is the probability that in the future we will experience a worst performance than the observed worst-case. Here, β is the probability that – assuming that ε is the correct area – we would not have observed a test case in the ε area during our n observations. For example, if we had more test results, these could or could not be lower than the observed worst-case. In any case, with more observations, we are able to: (i) tighten the bound (i.e., decrease ε), (ii) increase the confidence (i.e., decrease β), or (iii) do both things to a lesser extent. Without running additional tests, we can tighten the bound at the cost of losing confidence in it. Alternatively, we could loosen the bound and increase our confidence.

We highlight that the theory does *not* require any prior knowledge on the probability distribution of the performance metric (i.e., on the dashed line in Figure 2). This is the strength of scenario theory with respect to the traditional methods that require assumptions on this probability distribution (e.g., its variance being finite).

We also remark that the test cases have to be randomly generated (or taken from the execution of the software in different scenarios). This is what guarantees the probabilistic characterisation. It could be argued, in fact, that what is actually used is only the test case where the system exposed the worst behaviour, and therefore this one is the only test case of interest. But identifying the testing conditions that expose the worst case might be not be straightforward and could require a greater effort than running a number of randomly generated test cases. In other cases, instead, the worst-case performance could be a trivial, arbitrarily bad performance. For example, the worst case response time of a web service will be infinite if all the servers become unavailable. Differently, we ask instead the following question: given a number of tests we ran on the real system, what is the average response time that we can guarantee in 99% of the cases? We argue that this probabilistic characterisation of self-adaptive software is (i) simpler to achieve and, (ii) more interesting than its deterministic counterpart. Therefore, when taking the probabilistic approach, even though a new test might not change the worst-case bound, it is still valuable because it increases the reliability and confidence in the obtained bound.

This probabilistic characterisation of the guarantees specifically addresses the research challenge **CH1**. Our argument is that, since deterministic guarantees cannot be given for adaptive systems, we should aim for probabilistic ones. Within the choice of probabilistic guarantees we have then addressed the other two research challenges. In fact, we have showed how to apply scenario theory for quantifying the system performance and testing confidence. Respectively, ε quantifies the probabilistic bound on the performance (**CH2**), and β quantifies the testing adequacy (**CH3**).

5 EXPERIMENTS

This section aims at validating the proposed methodology. Our approach is designed to: (i) provide formal probabilistic guarantees

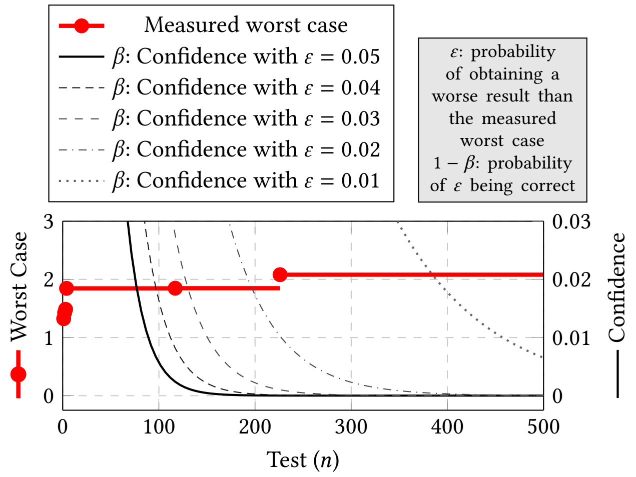

Figure 3: Measured worst case and confidence level varying the number of performed tests.

from experiments (**CH1**), (ii) allow us to perform a fair comparison of different adaptation strategies (**CH2**), (iii) quantify the trade-off between the number (and cost) of experiments and the obtained probabilistic confidence (**CH3**).

The proposed methodology is *application independent*. We highlight this strength presenting experimental data from well-established adaptive software with different application domains: healthcare and video processing. In particular, in Section 5.1 we show how the methodology exposes the trade-off between the number of performed tests and the obtained probabilistic confidence using a simulation tool for the Tele Assistance Service (TAS) [6, 70]. This shows how we have addressed the research challenges **CH1** and **CH3**. In Section 5.2 we discuss how the methodology can be used to compare different adaptation strategies using the Self-Adaptive Video Encoder (SAVE) [45]. Our approach allows us to address the research challenge **CH2**.

5.1 Data vs. Confidence Trade-Off

Aim: The aim of this set of experiments is to show the value of the proposed methodology in quantifying the trade-off between the number of performed tests and the testing confidence. We directly connect the amount of collected experimental data with the probabilistic testing confidence (**CH3**). The scenario theory offers guarantees on the software performance level, also for test cases that have not been explicitly executed (**CH1**).

Self-Adaptive Software: TAS is a service-oriented software application that provides care and assistance to elderly people that suffer from chronic diseases [6]. The software [70] periodically monitors patients conditions using sensors and activates a chain of services invocations. First, the patient conditions are sent to an *Analysis Service*, that inspects the data and determines the next steps to be taken for the patient well-being. The outcome of the analysis is one of the following: (i) do nothing, (ii) invoke a *Drug Service* that will compute a new medicine dosage, or (iii) invoke an *Alarm Service* that will dispatch an ambulance.

Each service can be realised by multiple service providers, potentially doing different computations that follow the same specification and interface. During the execution of the software, the selection of which provider to invoke to obtain a given functionality introduces an element of choice in the management of each request. Service providers have different properties; e.g., quality of the service, availability, success rate, and failure probability. In our experiments we focus on service rate and availability in the presence of failures, i.e., the number of requests processed per time unit and the probability of serving incoming request successfully.

The presence of different service providers and variety of potential needs for each request introduces the need to adapt the software behaviour to the current operating conditions. Adaptation strategies were introduced with the aim of selecting given services based on properties to be enforced for the overall system, e.g., [14, 24, 43, 60]. In our experiments, the adaptation strategy should recognise the service providers with higher service rates and prioritise them when distributing the requests. Also, since services might not always be available, the adaptation layer should avoid submitting requests to unavailable service providers.

To identify the best choices, the adaptation layer stores one number per service provider, called *weight*. For all the alternatives, the weight is initialised to 1 and incremented or decremented (using a fixed step equal to 50 in our experiments) based on the service performance. For each successfully processed request, the weight increases, and for each failed invocation the weight decreases. We further introduce a timeout and reset the weight to 1 if the service invocation failed for all the requests sent in the timeout interval.

When distributing the requests, the weights are used to define a probability distribution over the different providers of a given service. The probability distribution can be obtained by normalising each of the weights over their overall sum. The requests are distributed according to this probability distribution. We limit the weights to an interval between 1 and 1000. This avoids that overly positive weights attract all the requests. In the same way, negative weights imply that the service provider is never chosen, making it impossible to recover even in case of potentially correct operation.

Test Design: We use the TAS case study to highlight the trade off between data and confidence, i.e., how the exploration of the system's behaviour improves with the increasing number of tests. The definition of which inputs should be randomised is critical for the correct coverage of the system's behaviour. Here, we randomise: (i) the requests profile, i.e., the number of incoming requests; (ii) the workload mix, i.e., the type of incoming requests; (iii) the availability of the different service providers, i.e., a provider being reachable or not; and (iv) the reliability of the service providers, i.e., request processing may fail due to internal reasons.

We can use one or more performance parameters, depending on the specific software and on what are the aspects that we want to test. The performance parameter should be representative of the behaviour of the adaptation layer. Practically, this means that it should enable the distinction of whether the adaptation layer worked well for the specific test case, or not. In the TAS case we want to build a system that is robust to the occurrence of failures. We choose as performance parameter the average number of attempts needed for a request to be correctly handled. Lower numbers indicate better adaptation, 1 being the best possible value (often not achievable).

Results: Figure 3 shows the evolution of our quantities of interest when we perform an increasing number of tests. In particular, it shows: (i) the worst experienced average number of attempts needed per request (using the left y-axis), and (ii) the confidence β in the test outcome for different values of ε (using the right y-axis).

In the figure, we highlight with circle markers the newly experienced worst cases. The worst case is monotonically increasing with the number of conducted experiments. For example, in test #226, the average number of attempts per request to complete the TAS cycle is 2.081. This is a new worst case, as the previously experienced value was 1.8479 (from test #117).

The probability of not performing a relevant test (i.e., a test that would lead to a different worst case) is monotonically decreasing with the number of performed experiments. Analogously, a higher number of test cases is leading to a higher test coverage. Despite an unchanged worst case, between tests #117 and test #226, our confidence in the experimental results grew (lower values of β).

Decreasing the value of ε means being more conservative with our evaluation. The non-solid lines show the confidence β with smaller values of ε (up to 1%). Many more experiments are needed to obtain the same level of confidence when a smaller ε is selected.

Finally, using the scenario theory, we can state:

> *Based on the results of* $n = 500$ *tests, requests sent to TAS (with the described adaptation strategy) will not need more than* 2.081 *attempts on average to complete (despite service failures) with probability* $1 - \varepsilon = 0.98$. *This statement is correct with probability* $1 - \beta = 0.99996$.

This performance is apparently strongly dependant on the chosen adaptation strategy. More interestingly, it does not depend on the specific values of the quantities that have been randomised for the test case generation. Conversely, we could determine the number of tests to be performed based on the desired ε and β values:

> *Given the desired probabilistic guarantees of confidence of* $1 - \beta = 0.99996$ *and a bound that holds in* 98% *of the cases, we perform* $n = 500$ *tests. In our case, the* 500 *tests indicate that in the worst case* 2.081 *attempts are needed on average per request.*

Suppose that we could only afford to conduct $n = 250$ tests. In Figure 3 we can see that the measured worst case is the same as the complete test campaign. However, keeping $1 - \varepsilon = 0.98$, we could only claim a lower confidence in our test findings:

> *Based on the results of* $n = 250$ *tests, requests sent to TAS (with the described adaptation strategy) will not need more than* 2.081 *attempts on average to complete (despite service failures) with probability* $1 - \varepsilon = 0.98$. *This statement is correct with probability* $1 - \beta = 0.9936$.

Vice versa, we could also determine the larger bound ε that we need to accept for if we wanted the same confidence $1 - \beta = 0.99996$ for 250 experiments. In this case we would obtain $1 - \varepsilon = 0.9603$.

5.2 Adaptation Strategies Comparison

Aim: The aim of this second set of experiments is to show the use of the proposed methodology for the comparison of different adaptation strategies. We run the tests and quantify the performance for each case (**CH2**). The formal quantification allows us to compare in a fair way the different proposals. Moreover, we also show the application of the scenario theory for testing with different and conflicting adaptation requirements (**CH1**). To further emphasise the validity of the proposed methodology, in this section we run the tests using the real software, rather than a simulation tool.

Self-Adaptive Software: SAVE [45] is a video encoding tool that aims at automatically achieving the desired size compression of a video stream whilst preserving as much as possible of its content. We target video broadcasting services, where multiple videos are streamed with a fixed amount of bandwidth and unpredictable demands. We also assume that the video content is not known a priori and is expected to change over time. The need for adaptation arises from the strong dependence of the encoding performance on the specific content of the video.

The adaptation strategy should leverage the frame characteristics to autonomously find an effective combination of encoding parameters. For each frame, the adaptation layer selects: (i) the *quality* parameter that specifies the compression density. It ranges between 1 and 100, where 100 preserves all frame details and 1 produces the highest compression; (ii) the *sharpen* parameter, which specifies the size of a sharpening filter to be applied to the image. The filter size ranges between 0 and 5 where 0 indicates no sharpening; (iii) *noise correction*, which specifies the size of a noise reduction filter, also between 0 and 5. High filtering should in general generate a more uniform image, making it simpler to compress.

For each frame the adaptation layer measures size and quality and selects the encoding parameters accordingly, using its own algorithm. The size is measured in bytes and the quality is measured using the Structural Similarity (SSIM) index [74]. This index is a unitless metric that ranges between 0 and 1 and quantifies the similarity between the original and the encoded frame (high index meaning high similarity). The measurements are used to evaluate the *size error* and the *SSIM error* as differences between the measured values and the desired ones.

We compare four different adaptation strategies, two from the original artifact [45] and two developed specifically for this work:

- **Random**: this adaptation strategy (from the original artifact) selects random encoding parameters. We use it as a baseline for our evaluation.
- **Model Predictive Control (MPC)**: this adaptation strategy (from the original artifact) exploits model predictive control algorithm [32]. It solves a model-based optimisation problem for each frame and uses the result to determine the encoding parameters for the next frame. For our tests, we used the tuning parameters from the original publication [44].
- **Integral**: we developed an heuristic adaptation strategy, inspired by control theory principles. Here, the size error is used to choose the quality parameter. If the size is larger than the desired one, the quality parameter is reduced by 5. If smaller, the quality is increased by 5. The SSIM index determines the choice of noise and sharpen filter radius. Both

are increased by 1 if the quality is more than desired, and reduced otherwise. From an analytical perspective, the errors are *integrated* to perfect the encoding parameters choice.

- ϵ-**Greedy**: this adaptation strategy is based on the homonym machine-learning algorithm [63]. It alternatively leverages two adaptation approaches: (i) a *greedy* approach that exploits the knowledge of the best parameters already encountered with probability $1-\epsilon$, and (ii) a random approach that explores new possible choices, by randomly selecting new parameters with ϵ probability. The performance of a given choice of parameters is quantified based on the errors and normalised by the desired values. Higher similarity and lower size are desired, inducing errors that are close to zero. The greedy approach chooses the set of parameters that is associated to the lowest performance value. We use $\epsilon = 0.2$.

Test Design: In SAVE, adaptation takes place along a stream of frames, i.e. the feedback from one frame is used to improve the encoding of the next frame. To capture the behaviour of the adaptation strategy, each test should be an adequately long video, in which changes occur, triggering the need for adaptation. We would like to evaluate the performance of the different adaptation strategies independently from the content of the processed videos.

According to the proposed methodology, we define a set of videos that can be considered a random sample, with respect to their content. Here, we used the *User Generated Content* dataset from Youtube [67]. This dataset is representative of videos uploaded by users to Youtube. The videos are classified in categories and we focused on the sport category, because, due to the ever-changing scene, these are usually the most difficult to encode for real-time streaming and will expose the most of the adaptation strategy properties. The database contains 160 sport videos.

The adaptation strategy tries to achieve multiple objectives (a given size of the encoded frames, and a given content loss) at the same time. To capture the results obtained for both objectives, we define two different performance parameters, used to measure the outcome of the tests. The encoding performance on a single frame is directly quantified as the errors on: (i) the encoding size and (ii) the SSIM. For performance evaluation, we only consider relevant the cases in which the size is larger than the desired value or the quality is lower than the setpoint.

Intuitively, the size error is a problem when the images require more bytes than desired, and the SSIM quality is a problem when the image has less information than desired. We therefore evaluate the performance over a video of an adaptation strategy as the average of the size and SSIM errors weighted with the REctified Linear Unit, $relu(\cdot)$ function. The $relu(\cdot)$ function returns 0 for negative inputs and leaves the input unchanged for positive values. The complete formula for the performance parameters is shown in Equation (3), where $SSIM_v$ and $SIZE_v$ are the integrated errors on the video v, $SSIM_{sp}$ and $SIZE_{sp}$ are respectively the SSIM and size setpoints, $SSIM_i$ and $SIZE_i$ are the SSIM and size of the i-th frame and n_f is the number of frames in the video.

$$\begin{aligned} SSIM_v &= (1/n_f) \cdot \textstyle\sum_i relu(SSIM_{sp} - SSIM_i), \\ SIZE_v &= (1/n_f) \cdot \textstyle\sum_i relu(SIZE_i - SIZE_{sp}). \end{aligned} \tag{3}$$

In our evaluation, we use a SSIM reference of 0.9, preserving most of the content in the videos, and a frame size reference of 70% of the size of a frame randomly picked from the uncompressed video. The choice of having per-video references for the size is driven by the strong dependence of the frame size on the specific video.

Results: We ran the 160 encoding tests with each adaptation strategy. For each video v, we computed the two performance parameters $SSIM_v$ and $SIZE_v$ defined in Equation (3). The histograms in Figure 4 show the results of the tests.[4] The dashed grey lines mark the average performance for both similarity index and size, and the purple dotted lines highlight the worst case experienced during the tests. Tables 1 and 2 respectively show the performance parameters for the SSIM and frame size.

The number of performed tests $n = 160$ allows for the scenario parameters $\varepsilon = 0.03$ and $\beta = 0.008$. As for the TAS case study, this is not the only possible choice and a tighter bound could be traded for lower confidence (e.g. $\varepsilon = 0.01$ and $\beta = 0.04$) or vice versa (e.g. $\varepsilon = 0.05$ and $\beta = 0.0003$). Apparently, the two quantities hold equally for each of the tested adaptation strategies.

For the size performance, the MPC adaptation strategy vastly outperforms all the other strategies. This is achieved at the price of a SSIM adaptation performing worse than the Random strategy – i.e. the baseline. This is consistent with the adaptation objectives stated in the design of the strategy, where the size compression was considered the main objective [44]. The Integral adaptation achieves the complementary result with respect to the MPC strategy. It presents good performance (among the strategies studied here) from the point of view of the SSIM but exposes the worse performance for what concerns the size. This can be attributed to the decoupled approach between the adaptation objectives pursued with this adaptation. Size and quality are not really decoupled (although the adaptation strategy treats them as such) and cannot effectively be treated separately. The machine-learning based approach, ϵ-greedy, achieves good performance for both parameters. The SSIM performance is comparable to the one of the Integral adaptation and the size performance is in the order of the tens of kilobytes. This latter performance parameter can be considered small with respect the biggest frames in the dataset, whose size is a few gigabytes. The ϵ-Greedy adaptation strategy proves therefore to be the best one at simultaneously achieving both adaptation objectives. This can be attributed to the exploration of the possible combinations of encoding parameters and the coupled feedback used for the two objectives.

Our testing methodology guarantees that the comparison between the different adaptation strategies is fair, thanks to the rigorous quantification of the obtained bounds. In particular, for the ϵ-Greedy algorithm, the scenario theory ensures that with a probability of $1 - \varepsilon = 0.97$ we will not observe: (i) an error worse than 0.1777 for the SSIM performance parameter, and (ii) an error worse than 35191 Kb for the size performance parameter (see Equation 3 for the performance definitions). The confidence in our test campaign is of $1 - \beta = 0.992$. If there was need to tighten the bound or

[4]In the figure, we enforce the same scales for the axes to ease the comparison between the different plots. This results in hiding part of the plot of the size performance for the Integral strategy.

Table 1: SSIM performance [adimensional].

	Mean	Std	Max
Random	0.0710	±0.0054	0.3251
MPC	0.1145	±0.0068	0.4565
Integral	0.0315	±0.0029	0.1685
Greedy	0.0135	±0.0018	0.1777

Table 2: Size performance [bytes].

	Mean	Std	Max
Random	8806	±1033	82488
MPC	492	±94	8718
Integral	126373	±13942	992342
Greedy	1885	±318	35191

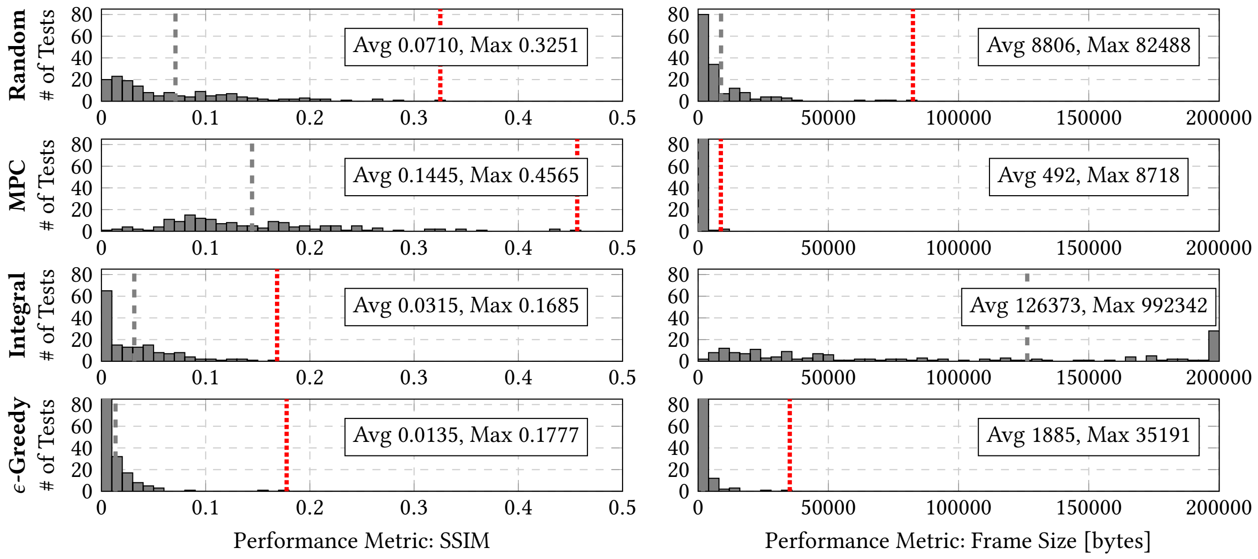

Figure 4: SAVE adaptation over the Youtube dataset with different techniques: Random, MPC, Integral, and ϵ-Greedy.

increase the confidence in the test campaign, the scenario theory would directly provide the extra number of test cases needed.

We close the discussion on the obtained result by highlighting the difference between worst-case and average-case metrics. Analysing the average case (as would be done for example by classical Monte Carlo approaches) for the results in Figure 4, one would conclude that the Random adaptation strategy actually performs more or less as well as the others. However, this is not at all true for the worst-case metrics, which clearly expose the trade-off between size and quality and the difference between having an adaptation strategy that targets one or both these quantities and picking the next frame configurations at random.

6 LIMITATIONS

The proposed approach has three main limitations. The first one is rooted in the definition of the testing of an adaptive system. The need for adaptation in a system rises from limited knowledge of the operational environment. This generates an intrinsic limitation to the definition of test cases, since the software, as a requirement, should adapt to new unforeseen circumstances. On the other side the testing process is only as effective as the test cases are representative of the real use case. These two objectives of the introduction of adaptation and rigorous definition of test cases are colliding [5]. The software engineer needs to synthesise a definition of the set of tests that adequately covers the adaptation use cases. However, the adaptive layer programmer has an interest in leaving the use cases as undefined as possible, for generality. In the TAS example, we would like the adaptation layer to handle general providers failures. However, this also means that (for proper testing) we need to define possible service failure patterns.

The second limitation arises from the interpretation of the performance parameters as random variables. This is the key to exploit random sampling and to leverage the different theories that are based on probability theory. Achieving unbiased random sampling can be challenging, especially when randomness cannot be quantified. The testing engineer must select a significant and relevant set of samples, e.g., sport videos with random content to test SAVE.

A last limitation arises from the need to conduct many tests to achieve high confidence. This is in fact time-consuming and the process needs to be automated. On the other side, the number of needed tests is known a priory and allows for timely allocation of the resources. Also, within scenario theory the confidence grows exponentially with respect the number of tests, avoiding the uncontrolled “explosion” of the number of tests to be executed.

7 CONCLUSIONS

In this paper we addressed the problem of testing the performance of a self-adaptive software system. Conventional testing techniques are limited in the guarantees they provide, due to the adaptation presence. The presence of adaptation makes this problem challenging, due to the need to test the system in the presence of uncertainty.

We proposed a probabilistic framework and leveraged the scenario theory, a tool from robust control that was originally intended for the design of control systems in the presence of uncertainty.

We reinterpreted the scenario theory results in light of our software testing problem. This allows us to provide formal probabilistic guarantees on the adaptation performance. Moreover, our method provides a probabilistic quantification of the testing adequacy, that can be used for the evaluation of testing coverage.

Finally, we empirically evaluated the effectiveness of our approach using two self-adaptive applications. We showed the trade-off between the experimental campaign volume and the confidence that can be obtained, and also demonstrated how to formally compare different adaptation strategies.

ACKNOWLEDGMENTS

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. The project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871259 (ADMORPH). The responsibility for the content remains with the authors.

REFERENCES

[1] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. 2012. *Learning From Data.* AMLBook.

[2] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. 2011. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press. https://books.google.se/books?id=zfIm94nNqPoC

[3] Andrea Arcuri and Lionel Briand. 2011. Adaptive Random Testing: An Illusion of Effectiveness? *(ISSTA '11).* ACM, New York, NY, USA, 265–275. https://doi.org/10.1145/2001420.2001452

[4] Andrea Arcuri, Gordon Fraser, and Juan Pablo Galeotti. 2014. Automated Unit Test Generation for Classes with Environment Dependencies *(ASE '14).* ACM, New York, NY, USA, 79–90. https://doi.org/10.1145/2642937.2642986

[5] R. I. Bahar, U. Karpuzcu, and S. Misailovic. 2019. Special Session: Does Approximation Make Testing Harder (or Easier)?. In *2019 IEEE 37th VLSI Test Symposium (VTS).* 1–9. https://doi.org/10.1109/VTS.2019.8758649

[6] L. Baresi, D. Bianculli, C. Ghezzi, S. Guinea, and P. Spoletini. 2007. Validation of web service compositions. *IET Software* 1, 6 (December 2007), 219–232. https://doi.org/10.1049/iet-sen:20070027

[7] Antonia Bertolino and Paola Inverardi. 2019. *Changing Software in a Changing World: How to Test in Presence of Variability, Adaptation and Evolution?* Springer International Publishing, Cham, 56–66. https://doi.org/10.1007/978-3-030-30985-5_5

[8] Antonia Bertolino, Paola Inverardi, and Henry Muccini. 2003. *Formal Methods in Testing Software Architectures.* Springer Berlin Heidelberg, Berlin, Heidelberg, 122–147. https://doi.org/10.1007/978-3-540-39800-4_7

[9] Marcel Böhme. 2019. Assurance in Software Testing: A Roadmap *(ICSE-NIER '19).* IEEE Press, Piscataway, NJ, USA, 5–8. https://doi.org/10.1109/ICSE-NIER.2019.00010

[10] Lionel Briand, Shiva Nejati, Mehrdad Sabetzadeh, and Domenico Bianculli. 2016. Testing the Untestable: Model Testing of Complex Software-intensive Systems *(ICSE '16).* ACM, New York, NY, USA, 789–792. https://doi.org/10.1145/2889160.2889212

[11] Giuseppe Carlo Calafiore. 2013. Direct data-driven portfolio optimization with guaranteed shortfall probability. *Automatica* 49, 2 (2013), 370 – 380. https://doi.org/10.1016/j.automatica.2012.11.012

[12] G. C. Calafiore and M. C. Campi. 2006. The scenario approach to robust control design. *IEEE Trans. Automat. Control* 51, 5 (May 2006), 742–753. https://doi.org/10.1109/TAC.2006.875041

[13] G. Canfora and M. Di Penta. 2006. Testing services and service-centric systems: challenges and opportunities. *IT Professional* 8, 2 (March 2006), 10–17. https://doi.org/10.1109/MITP.2006.51

[14] Mauro Caporuscio, Raffaela Mirandola, and Catia Trubiani. 2017. Building Design-time and Run-time Knowledge for QoS-based Component Assembly. *Softw. Pract. Exper.* 47, 12 (Dec. 2017), 1905–1922. https://doi.org/10.1002/spe.2502

[15] Francisco J. Cazorla, Tullio Vardanega, Eduardo Quiñones, and Jaume Abella. 2013. Upper-bounding Program Execution Time with Extreme Value Theory. In *13th International Workshop on Worst-Case Execution Time Analysis (OpenAccess Series in Informatics (OASIcs)),* Claire Maiza (Ed.), Vol. 30. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 64–76. https://doi.org/10.4230/OASIcs.WCET.2013.64

[16] Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, and T. H. Tse. 2010. Adaptive Random Testing: The ART of Test Case Diversity. *J. Syst. Softw.* 83, 1 (Jan. 2010), 60–66. https://doi.org/10.1016/j.jss.2009.02.022

[17] Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Giovanna Marzo Serugendo, Schahram Dustdar, Anthony Finkelstein, Cristina Gacek, Kurt Geihs, Vincenzo Grassi, Gabor Karsai, Holger M. Kienle, Jeff Kramer, Marin Litoiu, Sam Malek, Raffaela Mirandola, Hausi A. Müller, Sooyong Park, Mary Shaw, Matthias Tichy, Massimo Tivoli, Danny Weyns, and Jon Whittle. 2009. Software Engineering for Self-Adaptive Systems. In *Software Engineering for Self-Adaptive Systems: A Research Roadmap,* Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, and Jeff Magee (Eds.). Springer-Verlag, Berlin, Heidelberg, 1–26.

[18] Betty H. C. Cheng, Kerstin I. Eder, Martin Gogolla, Lars Grunske, Marin Litoiu, Hausi A. Müller, Patrizio Pelliccione, Anna Perini, Nauman A. Qureshi, Bernhard Rumpe, Daniel Schneider, Frank Trollmann, and Norha M. Villegas. 2014. *Using Models at Runtime to Address Assurance for Self-Adaptive Systems.* Springer International Publishing, Cham, 101–136. https://doi.org/10.1007/978-3-319-08915-7_4

[19] Laurens de Haan and Ana Ferreira. 2010. *Extreme Value Theory: An Introduction (Springer Series in Operations Research and Financial Engineering)* (1st edition. ed.). Springer.

[20] Vânia de Oliveira Neves, Antonia Bertolino, Gugliemo De Angelis, and Lina Garcés. 2018. Do We Need New Strategies for Testing Systems-of-systems? *(SESoS '18).* ACM, New York, NY, USA, 29–32. https://doi.org/10.1145/3194754.3194758

[21] Saikat Dutta, Owolabi Legunsen, Zixin Huang, and Sasa Misailovic. 2018. Testing Probabilistic Programming Systems *(ESEC/FSE 2018).* Association for Computing Machinery, New York, NY, USA, 574–586. https://doi.org/10.1145/3236024.3236057

[22] Saikat Dutta, Wenxian Zhang, Zixin Huang, and Sasa Misailovic. 2019. Storm: Program Reduction for Testing and Debugging Probabilistic Programming Systems *(ESEC/FSE 2019).* Association for Computing Machinery, New York, NY, USA, 729–739. https://doi.org/10.1145/3338906.3338972

[23] Nicolas D'Ippolito, Víctor Braberman, Jeff Kramer, Jeff Magee, Daniel Sykes, and Sebastian Uchitel. 2014. Hope for the Best, Prepare for the Worst: Multi-Tier Control for Adaptive Systems *(ICSE 2014).* Association for Computing Machinery, New York, NY, USA, 688–699. https://doi.org/10.1145/2568225.2568264

[24] Ross Edwards and Nelly Bencomo. 2018. DeSiRE: Further Understanding Nuances of Degrees of Satisfaction of Non-functional Requirements Trade-off *(SEAMS '18).* ACM, New York, NY, USA, 12–18. https://doi.org/10.1145/3194133.3194142

[25] Paul Embrechts. 2000. Extreme Value Theory: Potential And Limitations As An Integrated Risk Management Tool. *Derivatives Use, Trading and Regulation* 6 (02 2000).

[26] Paul Embrechts, Thomas Mikosch, and Claudia Klüppelberg. 1997. *Modelling Extremal Events: For Insurance and Finance.* Springer-Verlag, Berlin, Heidelberg.

[27] Fabiano Cutigi Ferrari, Joost Noppen, Ruzanna Chitchyan, and Awais Rashid Lancaster. 2011. Investigating Testing Approaches for Dynamically Adaptive Systems Work in Progress.

[28] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. 2011. Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011).* IEEE, Lawrence, KS, USA, 283–292. https://doi.org/10.1109/ASE.2011.6100064

[29] Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2014. Automated Design of Self-adaptive Software with Control-theoretical Formal Guarantees *(ICSE).* ACM, New York, NY, USA, 299–310. https://doi.org/10.1145/2568225.2568272

[30] R.A. Fisher. 1930. *The Genetical Theory of Natural Selection.* OUP Oxford.

[31] B.A. Francis and P. Khargonekar. 1995. *Robust control theory.* Springer-Verlag. https://books.google.se/books?id=81vvAAAAMAAJ

[32] C. E. Garcia, D. M. Prett, and M. Morari. 1989. Model Predictive Control: Theory and Practice&Mdash;a Survey. *Automatica* 25, 3 (May 1989), 335–348. https://doi.org/10.1016/0005-1098(89)90002-2

[33] Carlos A. González, Mojtaba Varmazyar, Shiva Nejati, Lionel C. Briand, and Yago Isasi. 2018. Enabling Model Testing of Cyber-Physical Systems *(MODELS '18).* ACM, New York, NY, USA, 176–186. https://doi.org/10.1145/3239372.3239409

[34] Vincenzo Gulisano, Alessandro V. Papadopoulos, Yiannis Nikolakopoulos, Marina Papatriantafilou, and Philippas Tsigas. 2017. Performance Modeling of Stream Joins *(DEBS '17).* ACM, New York, NY, USA, 191–202. https://doi.org/10.1145/3093742.3093923

[35] Aymeric Hervieu, Benoit Baudry, and Arnaud Gotlieb. 2012. Managing Execution Environment Variability during Software Testing: An Industrial Experience. In *Testing Software and Systems,* Brian Nielsen and Carsten Weise (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 24–38. https://doi.org/10.1007/978-3-642-34691-0_4

[36] Robert M. Hierons and Mercedes G. Merayo. 2009. Mutation Testing from Probabilistic and Stochastic Finite State Machines. *J. Syst. Softw.* 82, 11 (Nov. 2009), 1804–1818. https://doi.org/10.1016/j.jss.2009.06.030

[37] San-Yih Hwang, Haojun Wang, Jian Tang, and Jaideep Srivastava. 2007. A Probabilistic Approach to Modeling and Estimating the QoS of Web-services-based Workflows. *Inf. Sci.* 177, 23 (Dec. 2007), 5484–5503. https://doi.org/10.1016/j.ins.2007.07.011

[38] J. Hänsel, T. Vogel, and H. Giese. 2015. A Testing Scheme for Self-Adaptive Software Systems with Architectural Runtime Models. In *2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*. 134–139. https://doi.org/10.1109/SASOW.2015.27

[39] Antonio Jiménez-Martín, Alfonso Mateos, and Sixto Ríos-Insua. 2005. Monte Carlo Simulation Techniques in a Decision Support System for Group Decision Making. *Group Decision and Negotiation* 14 (01 2005), 109–130. https://doi.org/10.1007/s10726-005-2406-9

[40] O. Johnson. 2004. *Information Theory and the Central Limit Theorem.* Imperial College Press. https://books.google.se/books?id=r5XI8a0lYykC

[41] Keyur Joshi, Vimuth Fernando, and Sasa Misailovic. 2019. Statistical Algorithmic Profiling for Randomized Approximate Programs *(ICSE '19)*. IEEE Press, 608–618. https://doi.org/10.1109/ICSE.2019.00071

[42] Brian Korver. 1994. The Monte Carlo Method and Software Reliability Theory.

[43] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. 2017. Automated Control of Multiple Software Goals Using Multiple Actuators *(ESEC/FSE 2017)*. ACM, New York, NY, USA, 373–384. https://doi.org/10.1145/3106237.3106247

[44] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. 2017. Automated Control of Multiple Software Goals Using Multiple Actuators *(ESEC/FSE 2017)*. ACM, New York, NY, USA, 373–384. https://doi.org/10.1145/3106237.3106247

[45] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. 2017. Self-adaptive Video Encoder: Comparison of Multiple Adaptation Strategies Made Simple *(SEAMS '17)*. IEEE Press, Piscataway, NJ, USA, 123–128. https://doi.org/10.1109/SEAMS.2017.16

[46] M. A. Mehmood, M. N. A. Khan, and W. Afzal. 2018. Automating Test Data Generation for Testing Context-Aware Applications. In *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*. 104–108. https://doi.org/10.1109/ICSESS.2018.8663920

[47] Zoltán Micskei, Zoltán Szatmári, János Oláh, and István Majzik. 2012. A Concept for Testing Robustness and Safety of the Context-Aware Behaviour of Autonomous Systems *(KES-AMSTA'12)*. Springer-Verlag, Berlin, Heidelberg, 504–513. https://doi.org/10.1007/978-3-642-30947-2_55

[48] Gabriel A. Moreno, Javier Cámara, David Garlan, and Bradley Schmerl. 2015. Proactive Self-adaptation Under Uncertainty: A Probabilistic Model Checking Approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, Bergamo, Italy, 1–12. https://doi.org/10.1145/2786805.2786853

[49] Gabriel A. Moreno, Alessandro V. Papadopoulos, Konstantinos Angelopoulos, Javier Cámara, and Bradley Schmerl. 2017. Comparing Model-based Predictive Approaches to Self-adaptation: CobRA and PLA *(SEAMS '17)*. IEEE Press, Piscataway, NJ, USA, 42–53. https://doi.org/10.1109/SEAMS.2017.2

[50] Freddy Munoz and Benoit Baudry. 2009. Artificial table testing dynamically adaptive systems. *CoRR* abs/0903.0914 (2009). arXiv:0903.0914 http://arxiv.org/abs/0903.0914

[51] Yi Qin, Chang Xu, Ping Yu, and Jian Lu. 2016. SIT: Sampling-based interactive testing for self-adaptive apps. *Journal of Systems and Software* 120 (2016), 70 – 88. https://doi.org/10.1016/j.jss.2016.07.002

[52] Federico Alessandro Ramponi and Marco C. Campi. 2018. Expected shortfall: Heuristics and certificates. *European Journal of Operational Research* 267, 3 (2018), 1003 – 1013. https://doi.org/10.1016/j.ejor.2017.11.022

[53] A. Reichstaller and A. Knapp. 2018. Risk-Based Testing of Self-Adaptive Systems Using Run-Time Predictions. In *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. 80–89. https://doi.org/10.1109/SASO.2018.00019

[54] Christian P. Robert and George Casella. 2005. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag, Berlin, Heidelberg.

[55] Christian P. Robert and George Casella. 2010. *Monte Carlo Optimization.* Springer New York, New York, NY, 125–165. https://doi.org/10.1007/978-1-4419-1576-4_5

[56] S. Rosario, A. Benveniste, S. Haar, and C. Jard. 2008. Probabilistic QoS and Soft Contracts for Transaction-Based Web Services Orchestrations. *IEEE Transactions on Services Computing* 1, 4 (Oct 2008), 187–200. https://doi.org/10.1109/TSC.2008.17

[57] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-Adaptive Software: Landscape and Research Challenges. *ACM Trans. Auton. Adapt. Syst.* 4, 2, Article Article 14 (May 2009), 42 pages. https://doi.org/10.1145/1516533.1516538

[58] L. Santinelli, Jérôme Morio, Guillaume Dufour, and Damien Jacquemart. 2014. On the Sustainability of the Extreme Value Theory for WCET Estimation. *OpenAccess Series in Informatics* 39. https://doi.org/10.4230/OASIcs.WCET.2014.21

[59] Ismayle de Sousa Santos. 2017. *TESTDAS: Testing MEthod for Dynamically Adaptive Systems.* Ph.D. Dissertation. Fortaleza, Brazil. Advisor(s) Castro Andrade, Rossana Mariade.

[60] Stepan Shevtsov and Danny Weyns. 2016. Keep It SIMPLEX: Satisfying Multiple Goals with Guarantees in Control-based Self-adaptive Systems *(FSE 2016)*. ACM, New York, NY, USA, 229–241. https://doi.org/10.1145/2950290.2950301

[61] Harnam Singh and Preet Pal. 2013. Software Reliability Testing using Monte Carlo Methods. *International Journal of Computer Applications* 69 (05 2013), 41–44. https://doi.org/10.5120/11834-7554

[62] Bento Rafael Siqueira, Fabiano Cutigi Ferrari, Marcel Akira Serikawa, Ricardo Menotti, and Valter Vieira de Camargo. 2016. Characterisation of Challenges for Testing of Adaptive Systems *(SAST)*. Association for Computing Machinery, New York, NY, USA, Article Article 11, 10 pages. https://doi.org/10.1145/2993288.2993294

[63] Richard S. Sutton and Andrew G. Barto. 1998. *Introduction to Reinforcement Learning* (1st ed.). MIT Press, Cambridge, MA, USA.

[64] Porfirio Tramontana, Domenico Amalfitano, Nicola Amatucci, Atif Memon, and Anna Rita Fasolino. 2019. Developing and Evaluating Objective Termination Criteria for Random Testing. *ACM Trans. Softw. Eng. Methodol.* 28, 3, Article 17 (July 2019), 52 pages. https://doi.org/10.1145/3339836

[65] T. H. Tse, Sik-Sang Yau, W. K. Chan, Heng Lu, and T. Y. Chen. 2004. Testing context-sensitive middleware-based software applications. In *Proceedings - International Computer Software and Applications Conference*, Vol. 1. 458–466.

[66] Huai Wang, W. K. Chan, and T. H. Tse. 2014. Improving the Effectiveness of Testing Pervasive Software via Context Diversity. *ACM Trans. Auton. Adapt. Syst.* 9, 2, Article Article 9 (July 2014), 28 pages. https://doi.org/10.1145/2620000

[67] Yilin Wang, Sasi Inguva, and Balu Adsumilli. 2019. YouTube UGC Dataset for Video Compression Research. arXiv:cs.MM/1904.06457 https://media.withyoutube.com/

[68] K. Welsh and P. Sawyer. 2010. Managing Testing Complexity in Dynamically Adaptive Systems: A Model-Driven Approach. In *2010 Third International Conference on Software Testing, Verification, and Validation Workshops*. 290–298. https://doi.org/10.1109/ICSTW.2010.57

[69] Danny Weyns. 2012. Towards an Integrated Approach for Validating Qualities of Self-Adaptive Systems *(WODA 2012)*. Association for Computing Machinery, New York, NY, USA, 24–29. https://doi.org/10.1145/2338966.2336803

[70] Danny Weyns and Radu Calinescu. 2015. Tele Assistance: A Self-adaptive Service-based System Examplar *(SEAMS '15)*. IEEE Press, Piscataway, NJ, USA, 88–92. http://dl.acm.org/citation.cfm?id=2821357.2821373

[71] Kohsuke Yatoh, Kazunori Sakamoto, Fuyuki Ishikawa, and Shinichi Honiden. 2015. Feedback-Controlled Random Test Generation *(ISSTA 2015)*. Association for Computing Machinery, New York, NY, USA, 316–326. https://doi.org/10.1145/2771783.2771805

[72] L. Yu, W. T. Tsai, Y. Jiang, and J. Gao. 2014. Generating Test Cases for Context-Aware Applications Using Bigraphs. In *2014 Eighth International Conference on Software Security and Reliability (SERE)*. 137–146. https://doi.org/10.1109/SERE.2014.27

[73] Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler. 2019. The Fuzzing Book. In *The Fuzzing Book*. Saarland University. https://www.fuzzingbook.org/ Retrieved 2019-09-09 16:42:54+02:00.

[74] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (April 2004), 600–612. https://doi.org/10.1109/TIP.2003.819861