Benchmarking the Resilience of Self-Adaptive Software Systems: Perspectives and Challenges

Raquel Almeida  
CISUC, Department of Informatics Engineering  
University of Coimbra  
Coimbra, Portugal  
rrute@dei.uc.pt

Marco Vieira  
CISUC, Department of Informatics Engineering  
University of Coimbra  
Coimbra, Portugal  
mvieira@dei.uc.pt

ABSTRACT
Self-adaptive systems are widely recognized as the future of computer systems. Due to their dynamic and evolving nature, the characterization of self-adaptation and resilience attributes is of utmost importance, but also presents itself as a huge challenge. In fact, currently there is no practical way to characterize self-adaptation capabilities, especially when comparing alternative systems concerning resilience. In this position paper we discuss the problem of resilience benchmarking of self-adaptive software systems. We identify a set of key challenges and propose a roadmap to tackle those challenges. At the same time, we present some perspectives on the development of such a benchmark, taking Autonomic Database Management Systems (ADBMS) as an illustrative case.

Categories and Subject Descriptors
D.2.8 [Software Engineering]: Metrics.

General Terms
Experimentation, Measurement, Standardization.

Keywords
Self-adaptive; benchmarking; resilience; autonomic database management systems.

1. INTRODUCTION
Faced with highly complex, insecure and nearly unmanageable computing systems, resulting from the increasing heterogeneity, scale, dynamism and interconnectivity of software, services and networks, the Information Technology industry and researchers are turning more and more their attention to self-managing systems; systems that are capable of self-configuration, self-optimization, self-healing, and self-protection [5, 10]. The key idea is that complex, large-scale, heterogeneous computer systems should have autonomic properties (i.e., they should be able to independently take care of standard maintenance and optimization tasks). Self-* characteristics allow decreasing the dependence on human managers, but eventually impact performance, dependability and security [13]. More than that, such characteristics become decisive to the system’s ability to adhere to its specifications (including dependability and security attributes) in the presence of disturbances of all kinds, including faults, attacks, and changes in available resources or in service demand.

Considering that self-adaptive systems are capable of adjusting their behavior in response to their perception of the environment and the system itself [4], they should be able to adapt and evolve to new states or configurations that are deemed more advantageous in face of the context (both internal and external), in an effort to maintain actual system attribute values close to desired specifications. Autonomic systems are then expected to guarantee a “service delivery that can justifiably be trusted, when facing changes” [11] (i.e., they must be resilient).

This trend of computer systems with autonomic characteristics leads to new research challenges, namely: how to evaluate these systems? How to determine the effectiveness of the self-adaptation mechanisms? How to assess the impact of the self-* properties in the performance, dependability and security of the system? How resilient are these systems? Furthermore, the development of similar self-adaptive software systems (i.e., systems that implement similar functional requirements) emphasizes the need for approaches that allow users to compare alternatives.

Some of these research problems have not yet been systematically tackled, and no concrete and comprehensive methodology has been studied and developed for the evaluation and comparison of autonomic capabilities in computing systems. The goal of resilience benchmarking is to provide generic ways for characterizing and comparing the behavior of components and computer systems when subjected to changes (both internal and external), allowing for the quantification of resilience metrics [1]. In practice, benchmarking the resilience of autonomic systems requires evaluating their capability of (effective) self-adaptation, and its impact on the service provided (from the user’s perspective).

Computer benchmarking is primarily an experimental approach, and so its acceptability is largely based on two fundamental facets of the experimental method: 1) the ability to reproduce the observations and measurements, either on a deterministic or a statistical basis, and 2) the ability of generalizing the results through some form of inductive reasoning. The first aspect (ability to reproduce) gives confidence in the benchmark results and the second (ability to generalize) makes the benchmark results meaningful and useful beyond the specific set up used in the benchmark process. In practice, benchmark results are normally reproducible in a statistical basis. The necessary results generalization is inherently related to the representativeness of the benchmark experiments, which ultimately means that the conditions used to obtain the measures are representative of what can be found in the real world.

In this paper we discuss the problem of resilience benchmarking...
of self-adaptive software systems. We identify and discuss some key challenges in the development and validation of such benchmarks, considering the unique characteristics of the benchmarking targets. Additionally, we present some perspectives on the development of the fundamental components of a resilience benchmark for self-adaptive software systems, namely in what concerns the definition of useful metrics and representative workloads and changeloads, using Autonomic Database Management Systems (ADBMS) to illustrate some essential points. This work represents our first step towards the establishment of a methodology for the empirical evaluation and benchmarking of self-adaptive software systems (including validation criteria, experimental designs and procedures), which may represent a key step in achieving a widely accepted and standardized way to evaluate and compare this type of systems.

The outline of the paper is as follows. The next section introduces background concepts on resilience benchmarking. Section 3 discusses current research challenges on benchmarking the resilience of self-adaptive software systems. Section 4 proposes some perspectives on the design of the benchmark components. Finally, Section 5 concludes the paper.

2. BENCHMARKING RESILIENCE

The objective of any computer benchmark is to provide a practical way to characterize and compare systems or components according to specific characteristics (e.g., performance, dependability) [8]. A fundamental aspect that distinguishes benchmarking from other existing evaluation and validation techniques is that a benchmark represents a technical agreement (accepted both by the computer industry and/or by the user community), intending to provide a reproducible and cost-effective way of performing such a characterization, mainly for comparative purposes. A benchmark must be as representative as possible of a given domain but, as an abstraction of that domain, it will always be an incomplete representation. The objective is to find a useful representation that captures the essential elements of the domain and provides practical ways to characterize the features of interest [9]. Still, it should be stressed that the resulting characterization is relative, in the sense that it is valid for the conditions under which it is obtained, and is not meant to be an absolute measure or guarantee of how a specific system will perform under field conditions.

From a practical point of view, a benchmark is a specification of a procedure to assess measures related to some facet of the behavior of a computer system or component, and specifies the benchmarking target, the metrics used, the way and conditions under which measures are obtained, and their domain of validity. A useful benchmark must be representative, portable, repeatable, scalable, non-intrusive and simple to use, and its validation consists essentially in verifying that it fulfills these key properties. Previous work on computer benchmarking can be divided in three main areas: performance benchmarking, dependability benchmarking, and security benchmarking. Resilience benchmarking is the next evident step, and will understandably encompass constructs and techniques from these previous benchmarking efforts.

When addressed in computer systems contexts, resilience has been linked to the capacity of a system to endure disturbances and changes in its environment, and still maintain its expected performance, or as a more encompassing perception of fault-tolerance. A review of some of the contemporary definitions used for resilience is presented in [12] and [14]. In our work we follow the taxonomy presented in [1] and [11], where resilience is defined as "the persistence of dependability when facing changes", being dependability an integrating concept [3], encompassing diverse attributes that are distinct from (although closely related to) system security, which is defined as "absence of unauthorized access to, or handling of, system state".

In this perspective, we consider the resilience of a system as its ability to maintain service delivery that can justifiably be trusted in spite of changes in its internal and external contexts or in the interface between these (i.e., the avoidance of failures that are unacceptably frequent or severe, even when facing changes). Obviously, what is "acceptable" will be decisively dependent on the system considered and its usage environment, and different dependability attributes will accordingly have distinct weight and significance.

Given an application domain (that specifies the target systems and benchmarking context), a resilience benchmark should provide generic ways for characterizing a system behavior in the presence of changes, allowing for a quantitative comparison of similar systems. If a system is effective and efficient in accommodating or adjusting to changes, avoiding failures as much as possible, it is reasonable to consider it as being resilient. This capability can be benchmarked by submitting the system to various types of changes and by observing the failures (and their frequency), as well as time and resources dedicated to avoid/recover from them. Still, the changes that the system has to face may lead to performance and dependability attributes degradation without leading necessarily to catastrophic system failures. Thus, we need to assess variations of the properties of interest (e.g., performance, availability, integrity) when the system is under varying context conditions, in order to characterize its behavior from a resilience perspective.

As appropriately pointed out in [12], evaluating resilience must consider the system and environment dynamics that are beyond those typically addressed in the evaluation of dependability. Comparing to already established performance and dependability benchmarks, a resilience benchmark could be specified following the same basic structure, but would necessarily include two new elements, namely: a changeload and resilience metrics. Of course, such system and environment dynamics will also markedly impact the procedure and rules of the benchmark. Figure 2 depicts the common components of already existing performance and dependability benchmarks, and relates them to our proposal for the structure of a resilience benchmark.

While maintaining similar workloads, dependability benchmarks enhanced performance benchmarks by introducing a faultload and dependability metrics, which include performance metrics under faulty conditions [15]. A resilience benchmark must comprise a more wide-ranging changeload, which will certainly include (but will not be limited to) faults. For instance, variations on the workload or in system parameters should be part of a changeload. New metrics for characterizing resilience are also needed, although some will naturally be based on measures of performance and dependability while facing changes.

Due to the high diversity of software systems, the definition of a resilience benchmark that fits all types of systems is an unattainable goal. This way, a benchmark must be specifically targeted to a particular domain. Moreover, the division of the spectrum into well-defined areas is mandatory to make it possible to make
choices during the definition of the benchmark components (e.g., metrics, workload, procedure). In this work we take Autonomic Database Management Systems (ADBMS) as an illustrative case. Based on an experimental approach, the focus is the assessment of quantitative measures of system resilience, with the main purpose of comparing alternative systems according to their ability to maintain service quality and dependability, as perceived by the user’s point of view, when facing representative changes of variant nature, prospect and timing.

3. CHALLENGES ON BENCHMARKING RESILIENCE OF SELF-ADAPTIVE SOFTWARE SYSTEMS

In light of the concepts discussed in the previous section, benchmarking the resilience of a self-adaptive system should consist in assessing the resilience of the system in the different states that it evolves to, when subjected to representative changes. This includes assessing performance and dependability aspects in different combinations of (state, context) of the system (e.g., initial stable state and final stable state, before and after the introduction of a change), and understanding how those measures vary in the presence of change. Furthermore, it is important to assess the impact on the service provided due to the behavior of the system during adaptation (as it evolves from one state to another).

The same change may have different consequences, depending on the particular context of the system, and on the specific trajectory of adaptation the system has been through. So, subjecting a self-adaptive system to a specific change might lead to different states, with distinct resilience. We assume that the process of adapting to changes affects the overall resilience of the system, and so consider that a resilience benchmark must take into consideration not only the effects of submitting the system to a particular set of individual changes (as was the case with faults in dependability benchmarks), but also to representative sequences of changes. In other words, we need to evaluate not only the impact of a change in the performance and dependability of the system, but also the overall impact of the adaptation triggered by a given change in the capability to face a subsequent change (in practice, adaptation may be seen as a change in the system, which should also be considered from a benchmarking point-of-view).

Evaluating autonomic systems, and their capability to effectively and efficiently self-adapt as changes occur, should include not only functional behavior, but also non-functional properties such as response time, performance, reliability, efficiency and security, to gain an overview of how well they are exerting their self-* capabilities (see also Figure 3). We also need to keep in mind that restricting evaluation to performance and dependability attributes in face of changes is insufficient to characterize self-adaptive software systems concerning their resilience. A self-adaptive system will probably often need to perform trade-offs between potentially conflicting goals. An adaptation performed in face of a particular change may result in the relaxation of some self-protective features, or even on a decrease of some service level, but may still correspond to an overall more advantageous state of the system in the presence of that change.

After the injection of a change, the system may take a long time to converge to a new stable state, or not converge at all. The overhead of adaptation may cause the system to end up thrashing, if it decides to act on every change in the environment, or if changes that would normally be worth of acting upon are too frequent. Unnecessary adaptations, missed opportunities to adapt, and any difficulties of the system to avoid constant oscillations of behavior or to converge to a satisfactory (desirable) stable state should reflect on performance and dependability measures, and have an impact on the resilience evaluation. This has implications both on the model chosen for the procedure of the benchmark (especially in what concerns timeframes for the injection of changes to the system under benchmark, and decisions on measurement intervals) and on the computation of resilience measures.

The previous paragraphs highlight just a subset of challenges brought by the specific nature of the target systems of a resilience benchmark. However, many other challenges need to be addressed, including: how to define and validate metrics that characterize the resilience of self-adaptive systems? How to define representative changeloads? How to define dynamic workloads? How to apply them to the system under benchmark? How to synthesize results of performance and dependability measures of distinct states of the system, resulting from several different changes or sequences of changes (both in internal and external contexts), into an overall measure of resilience of the system?

Addressing the previous issues in a comprehensive way to achieve a benchmark that could be adapted to all types of self-adaptive software systems is an unattainable goal. Furthermore, what is
considered to be a self-adaptive software system varies greatly, and their specific classification according to the modeling dimensions (goals, change, mechanisms and effects [2]) can be quite diverse from one system to another, and undoubtedly should impact the choices made during the design of the various components of the benchmark. So, in order to make it feasible the design of a resilience benchmark, and tackle the challenges discussed previously, we need to narrow down the scope of the benchmark application target.

In this paper we focus on the definition of a resilience benchmark for self-adaptive software systems with static and constrained goals, but that may have diverse duration or degree of dependency. We consider both internal and external sources of change, but restrict to foreseen and foreseeable changes, with variable frequency. Concerning the other modeling dimensions, most facets will in fact be partly the target of our evaluation, with exception of the ones for which we cannot obtain information while regarding the system as a black box. Note that further assumptions may be needed, but these will only be identifiable when defining and validating the concrete benchmark.

4. PERSPECTIVES ON THE DESIGN OF THE BENCHMARK COMPONENTS

As introduced in Figure 2, currently we are considering that the main components of a resilience benchmark would be: metrics (to characterize the resilience of the system when submitted to the changeload), workload, changeload, the specification of the benchmark procedure and rules, and the experimental setup required to run the benchmark. In the following subsections we address some of the challenging issues on the design of the benchmark components. For illustration purposes, and to frame the research roadmap and our perspectives on the design of specific benchmark components, we consider Autonomic Database Management Systems (ADBMSs) as the benchmark target.

4.1 Metrics

Metrics should allow characterizing and quantifying the system behavior when facing changes (e.g., faults, changes in priority of goals, operational environment variations). Defining and validating metrics for expressing comprehensively and effectively the resilience level of a software self-adaptive system, in such way that allows comparison, is one of the greatest challenges of the design of a resilience benchmark.

In a recent work [7], the authors proposed an interesting set of criteria for the description and evaluation of the adaptive properties of autonomic systems, in an effort to provide concrete mechanisms to analyze the quality of design of adaptive systems, to evaluate the effect of self-* properties on performance, and to compare the adaptive features of different systems. Some of the proposed criteria are related to performance, robustness and self-stabilization evaluations of versions of a system with or without self-* characteristics, and could eventually be interesting in the design of a resilience benchmark for self-* systems. Nonetheless, they are directed at evaluating the impact of adding self-* characteristics to a system, rather than focusing on assessing the variations of behavior of an autonomic system when faced with changes at runtime, and thus are not suitable for the comparison of distinct self-adaptive systems that are offered as alternatives for the execution of comparable functions.

Our benchmark is intended to evaluate the resilience of a self-adaptive software system while executing a representative workload and being subjected to a representative changeload, and thus allow the comparison of similar systems in similar conditions, from a user’s perspective. To quantify resilience, we propose the use of performance and dependability attributes (such as availability, performability, etc.) and their variation as the system deals with changes. System performance and dependability attributes may eventually deviate from normal (or acceptable) behavior in the presence of each change, or sequence of changes, and those variations are, in fact, what will ultimately characterize the self-adaptive system resilience when submitted to a given changeload, in the specific context of the benchmark. Service-related metrics like performance (both baseline and in the presence of the changeload), up-time, and robustness seem sensible to be a part of the metrics set for such a benchmark. Time-related metrics are also important, as the ability to “guarantee service that can justifiably be trusted while facing changes” is obviously impacted by the time that the system takes to detect a change, decide if it is worth of acting upon, plan the action and executing it.

For a resilience benchmark for ADBMSs, the metrics set could include: performance metrics inherited from already established performance benchmarks (e.g., number of transactions executed per minute), obtained both in a stable state condition and while the system is dealing with a change; dependability metrics, such as uptime, or number of data integrity errors detected; and time-related metrics, such as time the system takes to adapt, or time it takes to stabilize its operation. These metrics would be obtained for each sequence of changes included in the changeload, and would then need to be combined in an overall resilience evaluation of the system. They would also need to be leveraged with the baseline values obtained for the system (when operating in the absence of changes) and the specific target values determined by the systems specific goals, in order to allow a trustworthy comparison of distinct systems concerning their ability to maintain expected behavior when facing changes.

4.2 Workload

During the benchmark execution, the system under test must be submitted to a representative set of tasks, which should be as close to real conditions as possible. So, the workload represents a typical operational profile for the considered area. An important aspect is that a workload for self-adaptive systems cannot be static and must include tasks that exercise the autonomic capabilities of the system (certain characteristics of the workload may lead to the need for adaptation), as the real conditions would.

Three different types of workload can be considered for benchmarking purposes [6]: real workloads (made of applications used in real environments, that have to face real changes), realistic workloads (artificial workloads that are based on real applications and changes in the domain of the benchmark), and synthetic workloads (e.g., randomly selected). While real workloads are representative, they are also difficult to obtain. On the other hand, synthetic workloads are easy to define, but their representativeness is doubtful. Realistic workloads stand in the middle and are artificially defined based on real workloads, in a way that guarantees an adequate degree of representativeness.

The workload for a resilience benchmark for ADBMSs may be based on already established workloads, such as those of the TPC performance benchmarks (http://www.tpc.org). However, when adopting an existing workload modifications may be required in order to target specific benchmarking requirements and system features. An important aspect to keep in mind is that the workload
itself should be dynamic as the goal is not to evaluate static performance but to assess specific resilience features.

4.3 Changeload

The definition of representative changeloads is probably the most difficult part of defining a resilience benchmark. A system may be subjected to distinct types of changes during its operation, and a resilience benchmark must try to emulate those as realistically as possible. Reducing the changeload to faults would be reducing the assessment to the fault-tolerance mechanisms or self-healing properties of the system: a resilience benchmark for self-adaptive systems must go well beyond that.

Although it is important to evaluate the ability of the system to maintain desirable behavior in the presence of failures of software or hardware components, it is equally relevant to test how the system behaves when facing other types of changes. These could include, for instance, the availability of new (more, or enhanced) software or hardware, resource exhaustion, load upsurges, faults introduced by operators, or even configuration changes.

Variations in the workload must also be part of the changeload. For ADBMSs, it is necessary to define and validate several modular workloads with distinct characteristics both in quantity (e.g., number of queries submitted per unit time, or number of client connections established) and in quality (e.g., type of queries submitted, where we can vary most accessed tables, or total number of aggregation functions used, for instance), in order to also exercise the self-tuning, self-optimizing capacities of these systems in presence of dynamic workloads.

Previously we discussed the importance of using representative sequences of changes, as the particular context of the system and its specific trajectory of adaptation may be determinant in terms of resilience. Figure 4 intends to help clarifying this issue, considering possible resulting states of the same system after being subjected to a same group of changes, but in a distinct order. As shown, States 1, 2, 3, etc., will probably have different characteristics and may (as consequence) also have distinct resilience. For instance, a system X could outperform another similar system Y using sequence (1), but the reverse situation could be verified using sequence (3). So, more than just identifying representative types of changes (e.g., through field studies and analysis), the definition and validation of a reference changeload must include the identification of representative sequences of the changes. Obviously, not all change sequences will be indispensable in the changeload: many may be very improbable, or in case of happening, have a minimal impact on the benchmarking targets. Risk analysis approaches may help in deciding which changes, or sequences of changes, are relevant enough to include in the changeload, taking into account both their likelihood and their expected impact. This way, our proposal on tackling the problem of the definition of a representative changeload comprises four sequential steps:

1. Identify and classify changes that the system may be subjected to;
2. Carefully study the set obtained in the previous step, and associate a probability of occurrence to each change;
3. Select from the previously obtained set of changes those that may be considered representative;
4. Determine relevant sequences of changes to use in a benchmark run.

Besides the problem of combining adequately several classes of changes, the representativeness issue is also directly related to the availability of practical means to apply the changeload. Still, we stress that the main problem is to obtain a realistic emulation of the impact and consequences of the changes in the system (e.g., the errors propagated, or the emerging of a need to adapt).

4.4 Benchmarking procedure

The benchmark execution could be divided in two different phases. The initial phase, or baseline phase, consists on a first run of the workload in a stable state, without submitting the system under benchmarking to injected changes. This phase is intended to determine the operational characteristics of the system while running the workload in the benchmarking context. The second phase is the test phase, to determine the operational characteristics of the system when submitted to the changeload, while running the workload. This latter phase is subdivided into a number of consecutive slots, as depicted in Figure 5.

Before submitting the system to any change, it is allowed to run at a steady state for a specific time (steady state time), to assure that the system is running correctly before any change activation. For an ADBMS, identifying the steady state condition would consist in executing the workload until the instant the system reaches the maximum number of transactions per minute.

For simplification of presentation, the measurement interval depicted in Figure 5 only includes the activation of a single change; but, as mentioned before, we intend to use sequences of changes, and so the phases included in this measurement interval could occur several times in each slot. The injection time will eventually depend on the type of change being introduced. After the activation of a change, the system will be left to try to deal with it, expectingly going through the four steps of a feedback control loop. The correspondent detection, analysis, planning and execution times are represented for illustration only, as our proposal is to benchmark the system as a black box (thus having no information of what is happening internally). Still, it would be interesting to have at least knowledge about the time to react and time to adapt, as these could be relevant in assessing the autonomy of the system, as well as deciding when operator recovery actions are needed. The final sub-phase is the keep time, intended to let the system ramp up and run at a steady state again.
gathering the agreement of all interested parties may be determinant for building trust between users and industrial communities, and boost the use of self-adaptive software systems.

6. REFERENCES