

Towards a Framework for Dependability Benchmarking

H. Madeira*, K. Kanoun**, J. Arlat**, D. Costa***, Y. Crouzet**, M. Dal Cin⁺, P. Gil⁺⁺ and N. Suri⁺⁺⁺

* University of Coimbra, Portugal ** LAAS-CNRS, Toulouse, France *** Critical Software, Coimbra, Portugal
⁺ FAU, Erlangen-Nürnberg, Germany ⁺⁺ UPV, Valencia, Spain ⁺⁺⁺ Chalmers University, Göteborg, Sweden

Abstract

The goal of dependability benchmarking is to provide generic ways for characterizing the behavior of components and computer systems in the presence of faults, allowing for the quantification of dependability measures. Beyond existing evaluation techniques, dependability benchmarking must provide a reproducible and cost-effective way of performing this evaluation either as stand alone assessment or more often for comparative evaluation across systems. This paper proposes a framework for defining dependability benchmarks for computer systems, with particular emphasis on COTS and COTS-based systems. The multiple dimensions of the problem are discussed and the framework is presented through a set of dependability benchmarking scenarios, ranging from experimental approaches to benchmarking scenarios where modeling and experimentation are tightly coupled. The main problems and proposals behind each benchmarking scenario are discussed and concrete case studies presented.

Keywords: Dependability Characterization, Dependability Assessment, Benchmarking, COTS components and COTS-based Systems.

Submission category: Regular paper

Word count: 6500

The material included in this paper has been cleared through authors' affiliations

Contact author:

Henrique Madeira
DEI-FCTUC
Polo II, University of Coimbra
3030 Coimbra
Portugal
Phone: (+351) 239 790003
Fax: (+351) 239 790003
Email: henrique@dei.uc.pt

1 Introduction

The goal of benchmarking the dependability of computer systems is to provide generic ways for characterizing their behavior in the presence of faults. Benchmarking must provide a reproducible way for dependability characterization. The key aspect that distinguishes benchmarking from existing evaluation and validation techniques is that a benchmark fundamentally should represent an agreement that is accepted by the computer industry and by the user community. This technical agreement should state the measures, the way and conditions under which these measures are obtained, and the validity domain. A benchmark must be as representative as possible of a domain. The objective is to provide practical ways to characterize the dependability of computers, so as to help system manufacturers and integrators improving their products as well as end-users in their purchase decisions.

The success of well-established performance benchmarks in comparing performance of components and computer systems probably accounts for the generalized idea that the main goal of benchmarks is to compare systems on the basis of benchmark results. However, we would like to emphasize that dependability benchmarks can be used in a variety of cases by both the end-users and manufacturers:

- *Characterize* the dependability of a component or a system.
- *Identify* weak parts of a system, requiring more attention and perhaps needing some improvements by tuning a component to enhance its dependability (e.g., using software wrappers), or by tuning the system architecture (e.g., adding fault tolerance) to ensure a suitable dependability level.
- *Compare* the dependability of alternative or competitive solutions according to one or several dependability attributes.

This paper proposes a framework for defining dependability benchmarks for computer systems, with emphasis on Commercial-Off-The-Shelf (COTS), and COTS-based systems, via experimentation and modeling.

Although a variety of evaluation techniques exist, aimed at different system levels and domains, a generalized approach that can be used to evaluate and compare different systems and applications still does not exist. Instead, several methods have been used, ranging from analytical modeling to simulation and experimental approaches, including those based on fault injection, e.g., see [1-7]. These approaches form a natural basis for our proposal for dependability benchmarking and the following paragraphs refer to those that are most appropriately related to dependability benchmarking.

It is worth mentioning the pioneering work that addressed the benchmarking of software robustness [8] and investigated the development of some forms of benchmarks [9, 10]. Among these studies, operating systems have received much attention, e.g., see [11-13] including development of tools (respectively, Ballista, Fuzz and MAFALDA) to support robustness testing. On the other hand, performance benchmarking is now a well-established area that is led by organizations such as the TPC (Transaction Processing Performance

Council) and SPEC (Standard Performance Evaluation Corporation), and supported by major companies in the computer industry, e.g., see [14].

Clearly, the preliminary efforts at developing dependability benchmarks, including focused robustness testing techniques, do not benefit yet from such a level of recognition. Much work is still needed despite the many relevant efforts that have tackled in a unified way several issues concerning performance and dependability assessment. Among these, we would like to point out the following studies that addressed jointly a couple of such issues: bringing together performance benchmarking and dependability assessment [15], field measurement and fault injection [16], field measurement and modeling [17], fault injection and modeling [18], and use of standard performance benchmarks as a workload for dependability evaluation [19]. To the best of our knowledge, little has been reported to cover these various facets in a comprehensive manner. Accordingly, this is the path that is specifically being explored in our work. This paper elaborates on our first thoughts reported in [20].

The ultimate objective of our work is to comprehensively define dependability benchmarks for computer systems, and provide means for implementing them. In this paper, we identify the main dimensions that are decisive for defining dependability benchmarks and show how they can be integrated into benchmarking scenarios to assess benchmark measures. These dimensions allow the description of i) the considered system in its operating environment, ii) the benchmark context, iii) the benchmark measures and iv) all what is needed for experimentation. A benchmark is defined when all of these dimensions are specified. Three examples of case studies are provided to exemplify how the dimensions of the proposed framework can be instantiated in real situations. The first one addresses operating systems that are the core components in computer systems. The other examples focus on transactional and embedded systems and applications.

It is expected that these case studies will aid defining the dependability benchmarks, at least for the considered classes of systems.

The paper is structured into five sections. Section 2 presents the relevant dimensions that characterize dependability benchmarking. Section 3 illustrates how the framework can be used for conducting dependability benchmarks. Section 4 describes the main specifications of preliminary benchmarks for three case studies. Finally, Section 5 concludes the paper.

2 Dependability Benchmarking Dimensions

The definition of a meaningful framework for dependability benchmarking requires first of all a clear understanding of all impacting dimensions. Their examination is essential to appreciate the problem space and to classify and specify all aspects of dependability benchmarks. Figure 1 outlines our classification of dimensions. The *categorization* dimensions allow us to organize the dependability benchmark space into well-identified categories. These dimensions describe the considered system and the benchmarking context. The *measure* dimensions specify the dependability benchmarking measure(s) to be assessed depending on

the choices made for the categorization dimensions. The *experimentation* dimensions include all aspects related to experimentation on the target system to get the base data needed to obtain the selected measure(s).

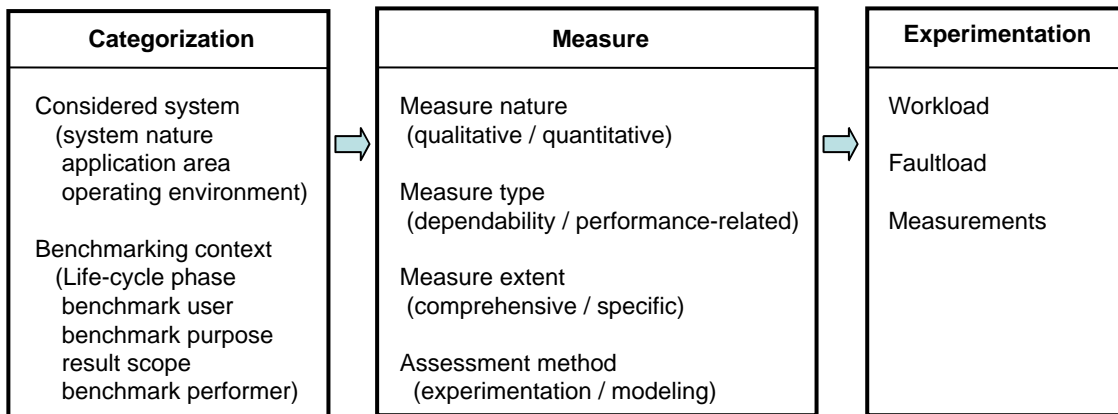


Figure 1 - Dependability benchmarking dimensions

In addition to these dimensions, a benchmark must fulfill a set of properties to be considered valid and useful. For example, it must be reproducible (in statistical terms), must not cause undesirable interferences or perturbations on the target system, must be portable, cost-effective, etc. These properties represent goals that must be achieved when defining actual dependability benchmarks.

In the following sub-sections we concentrate on detailing the three groups of dimensions as depicted in Figure1.

2.1 Categorization Dimensions

The categorization dimensions impact the basic selection of meaningful benchmark measures as well as the experimentation dimensions.

2.1.1 Considered System

It is essential to clearly identify the system under consideration, namely, its nature, its application area, as well as its operating environment. We detail each of them below.

System Nature - The system ranges from discrete hardware and software components (e.g., operating systems, middleware) to complete systems. Two conflicting views determine the way the system should be described in a dependability benchmark. On one hand, it would be desirable to know the system in detail to make possible the evaluation of specific measures such as measures directly related to fault tolerance mechanisms available in the system. On the other hand, a detailed description of the system could drastically reduce the portability of a benchmark. As a compromise, we propose two complementary views:

- Abstraction layer view: The target system is described in a generic way as a set of abstraction layers (or components) that perform specific functions, including key components for dependability. The main

question is how to reconcile the need of describing the target system to a given detail with the necessary level of portability required for a benchmark.

- **Functional view:** The system is described in terms of functionalities and features, including dependability features. The main question to be investigated is the extent to which this approach affects the accuracy of the various measures that can be obtained from the benchmarks. One possible way could be to associate, to some measures, disclosures specifying the detailed knowledge of the system required to obtain that measure.

Application Area - The application area is a key dimension for the definition of dependability benchmarks, as it impacts the system activation profile, the operating environment and the selection of benchmark measures. The division of the application spectrum into well-defined application areas is necessary to cope with the diversity of applications. The main difficulty is clearly the establishment of the appropriate granularity to divide the application spectrum. Obviously, different application areas will need different dependability benchmarks. For example, telecommunication applications have requirements that are different from those for control applications. The case studies presented in Section 4 illustrate this point.

Operating Environment - The operating environment characterizes the environment in which the system is used in operation. It includes aspects such as maintenance actions. As a consequence, it may affect the workload in a way that is usually ignored in performance workloads that concentrate essentially on functional aspects. Additionally many computer faults are induced by external sources that are intimately related to the operating environment. The main problem is to identify a way to take into account this environment. This is even more difficult when including human aspects. However, if the application area is well defined and bounded, it should be possible to define a typical operating environment to be used in experiments, including the set of operator faults that are agreed upon as typical for that application area.

2.1.2 Benchmarking Context

Performing a dependability benchmark and using the benchmark results can be done with multiple perspectives. The benchmarking context is, in fact, a composite dimension including the following:

Life Cycle Phase - A dependability benchmark could be performed in any phase of the system life cycle. The results could be used for the current or subsequent phases.

Benchmark user - Person or entity actually using the benchmark results.

Benchmark purpose - Results can be used either to characterize system dependability capabilities in a qualitative manner, to assess quantitatively these capabilities, to identify weak points or to compare alternative systems. The purpose may vary along the life-cycle, for example, during development, results could be used as a validation means or for identifying weak parts of the system and in operational life, they could be used to evaluate the impact of physical and design faults on system dependability.

Results scope - Benchmark results can be used internally or externally. External use involves standard results that fully comply with the benchmark specification, for public distribution, while internal use is mainly for system validation and tuning.

Benchmark performer - Person or entity performing the benchmark (e.g., manufacturer, integrator, third party, end-user). These entities have i) different visions of the target system, ii) distinct accessibility and observability levels for experimentation and iii) different expectations from the measures.

2.2 Measure Dimensions

Dependability benchmarks encompass measures assessed under particular conditions. Therefore an important task in benchmark definition concerns the identification of meaningful measures. Such measures allow the characterization of the system in a *qualitative manner* (in terms of features related to the system dependability capabilities and properties) or *quantitatively*.

As the occurrence or activation of faults may lead to performance degradation without leading to system failure, dependability and performance are strongly related. Thus the evaluation of system performance under faulty conditions (referred to as *performance-related measures*) allows completely characterizing the system behavior from a dependability perspective.

The various context perspectives impact the type of benchmark measures. Typically, end-users are interested in measures defined with respect to the expected services, referred to as *comprehensive measures*, while manufacturers and integrators could be more interested in *specific measures*.

Comprehensive measures characterize the system at the service delivery level, taking into account all events impacting its behavior. They may require the use of both experimentation and modeling. It is worth mentioning that this constitutes a new aspect in the context of computer benchmarking, as performance benchmarks rely only on direct measures.

A specific measure is usually associated with a specific system feature without necessarily taking into account all processes impacting its behavior (e.g., failure rates, maintenance duration). It may concern, for instance, error detection and recovery or fault diagnosis capabilities. For example, two specific measures can be associated to error detection: i) coverage of error detection mechanisms and ii) error latency (time required to detect an error). Such measures can be evaluated based on experimentation.

The actual set of benchmark measures is very dependent on the categorization dimensions. Moreover, more than one measure can be evaluated for a given set of categorization dimensions. Indeed a benchmark "measure" can be considered as a vector of measures, some of them being qualitative features, and some being quantitative measures.

2.3 Experimentation Dimensions

These include all aspects related to experimentation on the target system to obtain the selected measures, i.e., the workload, the faultload and measurements to be performed on the target system. Thus, these three dimensions are defined according to the categorization and measure dimensions.

2.3.1 Workload

The workload represents a typical operational profile for the considered application area. Widely accepted performance benchmarks can provide workloads for many application areas and clearly show that it is possible to agree on an abstract workload that reasonably represents a given application area. However, in dependability benchmarking, the concept of workload must be expanded to take into account the impact of preventive and corrective maintenance actions that are always conducted as part of routine operations.

2.3.2 Faultload

The faultload consists of a set of faults and exceptional conditions that are intended to emulate the real threats the system would experience. *Internal faults* (hardware or software faults) are mainly determined by the actual target system implementation. Considering an approach in which few assumptions are made on the actual target system structure (physical and logical), it is likely that the internal faults should be represented in the faultload by high-level classes of faults. In this context, equivalent sets of faults for different target systems must be understood on a statistical basis (used in the same application area) rather than being literally identical. *External faults* clearly depend on the operating environment.

The definition of the faultload will inevitably be a pragmatic process based on knowledge, observation, and reasoning. Inputs from many different application areas are needed to accomplish this task with success. Some examples are information from faults in the field (see e.g., [21] for software faults), characteristics of the operating environment (for operator faults), or even information from experimental and simulation studies.

The definition of representative faultloads is probably the most difficult part of defining a dependability benchmark. To make this problem tractable we are focusing, as a start, on well-defined application areas and considering faultloads that address only one class of faults at a time.

2.3.3 Measurements

Measurements performed on the target system allow the observation of the reaction of the system to the applied workload and faultload. The measures of interest are then obtained from processing these measurements.

2.4 Summary

This section allowed us to identify all dimensions required to define a dependability benchmark. The categorization dimensions define clearly the considered system (including its operating environment) and the benchmark context. Based on these dimensions, one or more benchmark measure(s) of interest can be defined. From an experimental point of view, the categorization and measure dimensions impact the three experimentation dimensions: the workload, the faultload and the measurements to be performed on the target system. Thus a central problem concerns the definition of suitable experimentation dimensions taking into account the other dimensions in an appropriate way.

3 Benchmark Conduct

A fundamental question entails ascertaining the final form of the product “dependability benchmark”? At this stage, giving the large spectrum of dimensions to take into account, we think that it is not reasonable to expect to be able to provide dependability benchmarks in a form that is “ready to use”, integrating all dimension choices consistently. As a result, the shape of dependability benchmarks could be a detailed specification including a list of procedures and rules to guide all the processes of *producing benchmark measures*, from experimentation and modeling. In particular, a dependability benchmark should include procedures and rules defining the way the specification should be implemented in the target system as well as procedures and rules for conducting experiments and to ensure uniform conditions for measurement.

We are working towards these objectives and we expect that the case studies considered in Section 4 will help us to define such specifications, at least for the considered classes of systems.

Assuming that such specifications do exist at least for some classes of systems, we now discuss the various steps required for conducting the dependability benchmark of a particular system or component.

Benchmarking is achieved in several steps forming a *benchmarking scenario*. Benchmarking starts by an *analysis step* that is necessary to settle on all dimensions (in many cases this step can be very simple). Characterization and assessment of the target system dependability is based on an *experimentation step* according to the selected dimensions. If comprehensive measures of the target system are of interest, a *modeling step* is required in addition to experimentation. Examples of modeling techniques for evaluating comprehensive measures are: block diagrams, faults trees, Markov chains or stochastic Petri nets.

Figure 2 gives the three steps for dependability benchmarking and their interrelations (links A to E). To illustrate how this general benchmarking framework can be used in real situations, we have chosen four examples of scenarios ranging from a purely experimental one to a more complete one in which modeling and experimentation are tightly coupled.

The rest of this section presents the three benchmarking steps and their interrelations, and introduces the four examples of benchmarking scenarios before discussing some related research issues.

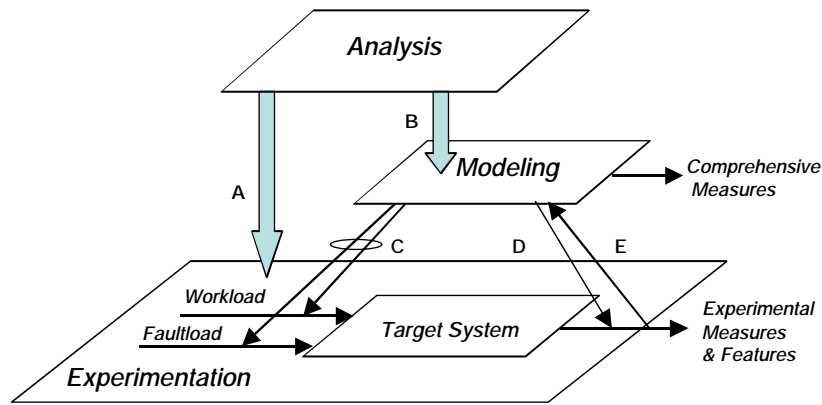


Figure 2 - Dependability benchmarking steps

3.1 Benchmarking Steps

The analysis step consists in defining the categorization and measure dimensions for the target system. If a benchmark corresponding to these dimensions is already available, the experimentation dimensions can be derived directly from this benchmark. Otherwise, they should be specified.

Two kinds of outputs are expected from the analysis depending on the benchmark measure assessment method. For experimental measures, outputs A in Figure 2 give the selected workload, faultload and measurements.

If modeling is required, the analysis step may, in addition, consist in analyzing deeply the system behavior to prepare modeling (outputs B). In which case, the selection of the workload, faultload and measurements can be guided by the modeling step as well.

The experimentation step pertains to carrying out the experimental part of the benchmark, i.e., executing the workload and faultload and collecting the response of the system as a result of measurements. The experimental measures are then obtained from the collected data.

However, as experimentation is tightly related to the target system, the workload and faultload defined at a high level during the analysis step or by the already existing benchmark specification, should be transformed or refined in order to be implemented at the target system level. Also, measurements are tightly related to the target system accessibility and observability points. Indeed, the benchmark should specify the kind of outputs required from the experiments to assess the benchmark measures. Deep examination of the target system allows determination of the accessibility and observability required for obtaining the specified outputs.

This refinement should be accomplished according to the procedures and rules included in the benchmark. In particular, these procedures and rules should address: scalability of the benchmark, configuration disclosures. Also, such rule should avoid "gaming", to produce biased results

Finally, it is worth mentioning that when the benchmark purpose is to identify system capabilities and possible weaknesses, no *numerical values* are to be estimated.

The modeling step lies in the description of the system behavior as well as interactions between components, taking into account failure and repair rates of system components. Processing of the analytical model to assess the measures requires the allocation of numerical values to the model parameters. Some of such values may be obtained from experimentation on the target system (e.g., coverage factors, restart time). Other parameter values can be provided from field data or based on past experience related to similar systems (e.g., failure and repair rates).

It is worth mentioning that models of large systems made of several components with several dependencies may be very complex (see e.g., [22]). However, for some systems, mainly COTS-based systems for which the details about the system architecture are not known, simple high-level models can be constructed. Such models may require only a reduced effort, thus offering a good trade-off for obtaining comprehensive measures characterizing globally the system with an acceptable effort.

Usually **experimentation and modeling** are used in a complementary way to assess system dependability [23]. Examples of relationships between modeling and experimentation are shown in Figure 2:

- Link C: Modeling is used to guide the selection of the most relevant classes of workload and faultload.
- Link D: Modeling is used to guide the selection of experimental measures and features. Actually, the constructed model is processed and sensitivity analyses with respect to parameter values are achieved to identify the most significant ones (i.e., associated to the most salient features of the system) that need to be accurately assessed by experimentation.
- Link E: The experimental results may lead to the refinement/correction of the analytical model. For example, the model may assume two failure modes, if experimentation shows the existence of a third one with a significant probability, the original model should be updated.

3.2 Benchmarking Scenarios

We have selected four examples of scenarios. The complexity of benchmarking and the associated effort increase from the simplest scenario (S1), based only on experimentation, to the most complete one (S4), in which modeling and experimentation are tightly coupled. However, more results can be obtained from the most complete one. Indeed, S1 is included in the three other scenarios. Hence, all the problems and research issues for S1 are also relevant for the others. These problems and research issues are briefly presented in Section 3.3.

S1: Dependability Benchmarking Based Only on Experimentation - The simplest scenario for benchmarking dependability follows a pure experimental approach. It includes analysis and experimentation steps and link A of Figure 2. It is actually an extension of the well-established performance benchmark setting. The analysis step specifies a set of measures and all the experiments necessary to obtain them. Typically, the target system is described in a functional way and few assumptions are made regarding its internal architecture. This approach can be used even when the documentation is not completely available as for COTS and COTS-based systems.

S2: Experimentation Supported by Modeling - S2 differs from S1 by the fact that experimentation is guided, at least partially, by modeling. It includes the three steps and links A, B, C and D. Modeling helps in selecting features and measures to be assessed experimentally. The outputs are similar to those of S1: experimental measures and features.

S3: Modeling Supported by Experimentation - S3 includes the three steps and links A, B and E. In S3, some specific experimental measures, used in the analytical model representing the system behavior, may be provided by experimentation. Moreover, system features identified by experimentation may impact the model semantics. Therefore, experimentation supports model validation and refinement. The outputs are comprehensive measures obtained from modeling. However, the experimental measures and features, assessed for supporting modeling, may be made available.

S4: Modeling and Experimentation S4 is a combination of S2 and S3. In this scenario, not only the selection of measures and features to be assessed experimentally are guided, at least partially, by modeling, but modeling is also supported by experimental results. Its outputs are experimental measures and features, as well as comprehensive measures based on modeling.

3.3 Some Research Issues

Representativeness is a crucial concern, as benchmark results must characterize the addressed aspects of the target system in a realistic way. Regarding established performance benchmarks, the problem is reduced to the representativeness of performance measures and of the workload. For dependability, it is also necessary to define representative dependability measures and representative faultloads. Although the problem seems clearly more complex than for performance benchmarks, the pragmatic approach used in the established performance benchmarks offers a basis for identifying adequate solutions for dependability benchmarking representativeness.

It is worth mentioning that many technical problems still need to be resolved. The subsequent points summarize crucial research issues.

- The adoption of the workloads of established performance benchmarks is the starting point for the definition of workloads. However, some work still has to be done. For example one has to check whether the way the application spectrum as partitioned by the performance benchmarks is adequate for this new class of dependability benchmarks.
- The definition of representative faultloads encompasses specific problems that are currently being studied (such as fault representativeness and the definition of practical way to apply the faultload to the system).
- Definition of meaningful measures. In particular, special attention should be paid to confidence and accuracy of measurements and the possible impact of the measurements on system behavior (intrusiveness).

4 Case Studies

In this section, we illustrate how the general concepts presented in the previous sections can be applied to examples of preliminary benchmarks. To study a representative set of benchmarks, thus allowing for the investigation of a substantial range of benchmark dimensions, we are considering three categories of target systems: i) *general purpose operating systems*, ii) *transactional systems* and iii) *embedded systems*. Concerning embedded systems, two classes of application domains are considered (space and automotive), whose implementations are respectively more software-oriented and hardware-oriented. In particular, the automotive application features a system-on-chip implementation.

To facilitate the cross-exploitation of the results and insights obtained when carrying out these studies, we chose to consider as much as possible a set of common benchmark categorization dimensions. Accordingly, hereafter, after describing the benchmark categorization dimensions, the emphasis will be put on the presentation of the most salient experimentation dimensions as well as of the measures.

4.1 Categorization dimensions

These dimensions are common to all case studies and are defined as follows:

- Target system: The target system is always either an *Off-the-Shelf* component, either commercial (COTS) or Open System Software (OSS) or a *system including at least one such component*.
- Life cycle phase: The benchmark is performed during the *integration phases*.
- Benchmark user: The primary users are the *integrators of the target system*. However, the benchmark results are to be communicated to the successive levels of integrators up to the *end-users* of the component or of the integrated system. Nevertheless, some results may be of interest to the developer(s) of the benchmarked OTS component, for improving the concerned component, in case the benchmark reveals some deficiencies.
- Benchmark purpose: For all target systems, the following possible purposes are identified: i) assess some dependability features, ii) assess dependability (and performance) related measures, and iii) identify potential weaknesses.
- Result scope: The benchmarks considered are primarily meant for *external use* (i.e., performed in a standardized way and made publicly available).
- Benchmark performer: The benchmark performer is either the *system integrator* or a *third party*. Indeed, we consider that the benchmark performer is someone (or an entity) who has no in depth knowledge about the component(s) being integrated and who is aiming at i) improving significantly its knowledge about the related dependability features, and ii) publicizing information on the dependability of the component and of the system under integration in a standardized way.

4.2 General Purpose Operating Systems

Operating Systems (OSs) form a generic software layer that provides basic services to the applications through an application-programming interface (API). Thus, they form privileged target software components for conducting dependability characterizations [24].

Two main approaches can be identified for benchmarking OS dependability, as viewed by the basic services a general purpose OS typically provide. First, a detailed analysis of candidate OSs can be carried out by decomposing them according to the set of common services they provide. Such decomposition can be mapped to a set of generic functional components that constitute the OS (e.g., synchronization, scheduling, memory management, driver controls and I/O functionality). It is very much suited for kernel components (especially for OSs based on microkernels (e.g., see [13]), but can be applied to a large extent to general purpose OSs, especially OSSs. An alternative way to perform a fair analysis can be obtained when the OSs exhibit a similar API. A typical example of such a popular API is the POSIX interface (e.g., see [11]).

The benchmark prototypes under investigation implement the dependability benchmark for general purpose OS in two well-known OSs: Linux and Windows 2000. They differ significantly in their internal architecture and in their implementation, which constitutes an excellent test of the benchmark portability. Additionally, the two approaches for OS benchmarking mentioned above are being investigated in these prototypes.

Workloads and Faultloads - As workload, we consider synthetic workloads respectively focused to the main functional components that have been identified above. These workloads correspond to specific programs aimed at activating the services provided by the functional components. For the faultloads, based on previous studies [25, 26], and as we are targeting COTS OSs, for which data on internal structure is usually not available, we concentrate on failure of the applications when interacting with the OS, by providing both invalid and valid erroneous parameters to the target OS.

Measures - At OS-level, we focus on the failure modes exhibited and that are characterized by the following events: i) kernel hang or crash, ii) exception and iii) error status report. Hindering (the error code returned is not correct) is also an interesting facet of the behavior [11]. It is important to note that these outcomes are not exclusive (several of such events can occur as the result of the application of a specific faultload). Thus, related timing observations are also of interest, as the order of occurrence of these events is important (e.g., the observation of an *exception* before a *kernel hang* is usually preferred to the opposite). Although precise timing measurements are not easy to achieve, especially in the case of a COTS OS (except for the case of exceptions), it might only be possible to observe the ordering of events, in this case.

It is worth noting that OS Reboot/Reset times resulting from the events listed above are also of interest. OS-specific timing measurements (such as switching time) should be more relevant for control applications than for transactional applications. Accordingly, it is worth pointing out that these specific analyses will be complemented by the application-specific benchmarks, as they also form the underlying OSs for the transactional and embedded systems considered in the next two sections.

4.3 Transactional Systems

We specifically address OLTP (On-Line Transaction Processing) systems, as they constitute the kernel of the information systems used to support the daily operations in most of businesses (banking, insurance, travel, health care, etc.). Such applications form traditional examples of business-critical areas, which clearly justify choosing them as targets to devise a dependability benchmark for transactional applications. The target system uses the Oracle database running on Windows 2000.

Workloads and Faultloads - We adopted the workload of the well-established TPC Benchmark™ C (TPC-C) [27]. It represents a mix of read-only and update intensive transactions that simulates the activities of most common OLTP environments, including transactions resulting from human operators working on interactive sessions. Concerning the faultloads, besides software faults (that can be emulated to some extent by machine-code level errors [28]), we emphasize also the analysis of the target system subjected to faults caused by human operators interacting with the system.

Measures - We naturally favor service-oriented measures that reflect the end-user point of view. Both dependability- and performance-related measures are being considered. The performance-related measure considered is the number of transactions executed per unit of time in the presence of a (standard) faultload. It measures the impact of faults on the performance. It favors systems with higher capability of tolerating faults, fast recovery time, etc. As is the case for performance benchmark, it might be useful to complement this measure with a cost-related measure (e.g., the price of processing transactions in the presence of a (standard) faultload). The main interest is to account for the benefit of including fault tolerance mechanisms in the system. It is worth noting that what matters here are not absolute performance figures, but rather the relative measures (e.g., degradation ratio due to the faultload).

In addition to such measures, it is possible to extend the observation to the behavior, in presence of faults, of particular system components. For example, the occurrence or activation of a fault may trigger the error detection mechanisms in the target system at various levels: hardware, OS, database engine and application. The fact that in the benchmark prototypes under investigation the transactions are processed on top of the same operating systems (i.e., Windows 2000 and Linux) used in the experiments with the benchmarks for general purpose OSs will provide further insights on the behavior of the OS in presence of faults. Other specific measures concern error detection and/or recovery latency and coverage. Finally, violations of properties (e.g., the well-known ACID properties: Atomicity Consistency, Isolation and Durability) provide another set of specific measures.

4.4 Embedded Systems

The two considered target systems are control applications from space and automotive industries. They both run on top of Linux. The study of two such target systems is meant to better account for the spectrum of work that could be performed for embedded systems. The first (software-oriented) system is a *control/navigation system for satellite* applications. The software was originally written in Ada language and

is based on the special *Thor* processor developed by Saab Ericsson Space [29]. It is being used in the *Odin* satellite that is currently operational. The system has been ported to the C language, using the POSIX standard. The second system features a system on-a-chip architecture for automotive application. The target system is a microcontroller (MPC565). The application is a subset of an electronic control unit for a Diesel engine. Two distinct architectures are being studied: a standard one featuring a single microcontroller, and a duplex architecture composed of two microcontrollers interconnected by a CAN network.

Workloads and Faultloads - The workloads are very much dependent of the applications running on each target system and thus will be tailored to these applications. For the space application, the faultload will be focused to test the robustness of the underlying OS at the level of the POSIX-compliant API. In the case of the system-on-a-chip target system, more emphasis will be put on simulating by means of bit-flips the consequences of hardware faults using the software implemented fault injection technique. The distribution and multiplicity of injected bit-flips will be based on results obtained from preliminary fault injection studies conducted using a VHDL simulator [30].

Measures - Here again, the main measures of interest concern the services offered by the respective applications. As both applications are real-time control applications, we are concerned with failures in the value domain (erroneous results) as well as on timing failures (such as deadline misses). This will allow us to extend the observations of the behavior of the OS in presence of faults, in particular to account for specific timing measurements. The latter include system call overhead, context-switch, etc. It is important to mention that the embedded applications context also needs to be considered to define the relevant measures accordingly. For the considered control applications, the additional (non-traditional) measures include aspects such as jitter, lag and stability that have ramification on the delivery of the services expected from the application. In addition, emphasis will be put on measures dedicated to assess the fault tolerance mechanisms featured by the systems, including hardware redundancy and exception handling mechanisms. Finally, the fact that both applications run on Linux will contribute to enhance the insights already obtained for this target OS. In particular, building upon the framework proposed in [7], observation means will be enhanced to allow for a detailed analysis of error propagation channels, mainly for internal purpose.

5 Conclusion

This paper was devoted to dependability benchmarks for computer systems, with particular emphasis on COTS components and COTS-based systems. Our goal is to determine generic ways for defining dependability benchmarks. Given the multi-dimensional nature of the problem, we started by the identification of all the dimensions of the problem. Three groups of dimensions have been identified: categorization, measure and experimentation dimensions.

The framework proposed for conducting dependability benchmarks encompasses three main steps. Benchmarking starts by an analysis step that is necessary to enable the choice of all dimensions. Characterization and assessment of the target system dependability is based on an experimentation step

according to the selected dimensions. If comprehensive measures of the target system are of interest, a modeling step is required in addition to experimentation. To illustrate how this framework can be used in real situations, we proposed four examples of benchmarking scenarios ranging from a purely experimental one to a more complete one in which modeling and experimentation are tightly coupled.

Finally, examples of dependability benchmarks concerning operating systems, transactional and embedded systems were presented. These benchmarks are still under study in the DBench project [31]. In particular, experiments will be set up to check the conceptual framework. More generally, a lot of work is still needed to validate the concepts introduced in this paper and to define dependability benchmarks in a comprehensive way. Due to the large spectrum of dependability benchmarking, we will concentrate our work essentially on the classes of target systems presented in this paper and expect to be able to define generic benchmarks at least for these classes of systems.

References

- [1] M. Malhotra and K. S. Trivedi, "Dependability Modeling Using Petri Nets", *IEEE Transactions on Reliability*, vol. 44, no. 3, pp. 428-440, September 1995.
- [2] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber and J. Reisinger, "Application of Three Physical Fault Injection Techniques to the Experimental Assessment of the MARS Architecture", in *Dependable Computing for Critical Applications (Proc. 5th IFIP Working Conf. on Dependable Computing for Critical Applications: DCCA-5, Urbana-Champaign, IL, USA, September 1995)*, (R. K. Iyer, M. Morganti, W. K. Fuchs and V. Gligor, Eds.), pp. 267-287, Los Vaqueros, CA, USA: IEEE Computer Society Press, 1998.
- [3] J. V. Carreira, D. Costa and J. G. Silva, "Fault Injection Spot-checks Computer System Dependability", *IEEE Spectrum*, vol. 36, 50-55, August 1999.
- [4] R. J. Martínez, P. J. Gil, G. Martín, C. Pérez and J. J. Serrano, "Experimental Validation of High-Speed Fault-Tolerant Systems Using Physical Fault Injection", in *Dependable Computing for Critical Applications (Proc. 7th IFIP Working Conf. on Dependable Computing for Critical Applications: DCCA-7, San Jose, CA, USA, January 1999)*, (C. B. Weinstock and J. Rushby, Eds.), pp. 249-265, San Jose, CA, USA: IEEE Computer Society Press, 1999.
- [5] M. Dal Cin, G. Huszerl and K. Kosmidis, "Quantitative Evaluation of Dependability-Critical Systems Based on Guarded Statechart Models", in *Proc. 4th Int. High-Assurance Systems Engineering Symp. (HASE-99)*, Washington, DC, USA, 1999, pp. 37-45 (IEEE CS Press).
- [6] J. Arlat, J. Boué and Y. Crouzet, "Validation-based Development of Dependable Systems", *IEEE Micro*, vol. 19, no. 4, pp. 66-79, July-August 1999.
- [7] M. Hiller, A. Jhumka and N. Suri, "An Approach for Analysing the Propagation of Data Errors in Software", in *Proc. Int. Conference on Dependable Systems and Networks (DSN-2001)*, Göteborg, Sweden, 2001, pp. 161-170 (IEEE CS Press).
- [8] A. Mukherjee and D. P. Siewiorek, "Measuring Software Dependability by Robustness Benchmarking", *IEEE Transactions of Software Engineering*, vol. 23, no. 6, pp., 1997.
- [9] T. K. Tsai, R. K. Iyer and D. Jewitt, "An Approach Towards Benchmarking of Fault-Tolerant Commercial Systems", in *Proc. 26th Int. Symp. on Fault-Tolerant Computing (FTCS-26)*, Sendai, Japan, 1996, pp. 314-323 (IEEE Computer Society Press).
- [10] A. Brown and D. A. Patterson, "Towards Availability Benchmarks: A Cases Study of Software RAID Systems", in *Proc. 2000 USENIX Annual Technical Conference*, San Diego, CA, USA, 2000 (USENIX Association).
- [11] P. Koopman and J. DeVale, "Comparing the Robustness of POSIX Operating Systems", in *Proc. 29th Int. Symp. on Fault-Tolerant Computing (FTCS-29)*, Madison, WI, USA, 1999, pp. 30-37 (IEEE CS Press).
- [12] J. E. Forrester and B. P. Miller, "An Empirical Study of the Robustness of Windows NT Applications Using Random Testing", in *Proc. 4th USENIX Windows System Symposium*, Seattle, WA, USA, 2000.

- [13] J. Arlat, J.-C. Fabre, M. Rodriguez and F. Salles, "Dependability of COTS Microkernel-Based Systems", *IEEE Transactions on Computers*, vol. 51, no. 2, pp. 138-163, February 2002.
- [14] J. Gray (Ed.) *The Benchmark Handbook for Database and Transaction Processing Systems*, San Francisco, CA, USA: Morgan Kaufmann Publishers, 1993.
- [15] D. Brock, "A Recommendation for High-Availability Options in TPC Benchmarks", <http://www.tpc.org/information/other/articles/ha.asp>
- [16] R. K. Iyer and D. Tang, "Experimental Analysis of Computer System Dependability", in *Fault-Tolerant Computer System Design*, (D. K. Pradhan, Ed.) pp. 282-392 (chapter 5), Upper Saddle River, NJ, USA: Prentice Hall PTR, 1996.
- [17] M. Kalyanakrishnam, Z. Kalbarczyk and R. K. Iyer, "Failure Data Analysis of LAN of Windows NT Based Computers", in *Proc. 18th Int. Symposium on Reliable Distributed Systems (SRDS'99)*, Lausanne, Switzerland, 1999, pp. 178-187 (IEEE Computer Society Press).
- [18] J. Arlat, A. Costes, Y. Crouzet, J.-C. Laprie and D. Powell, "Fault Injection and Dependability Evaluation of Fault-Tolerant Systems", *IEEE Transactions on Computers*, vol. 42, no. 8, pp. 913-923, August 1993.
- [19] D. Costa, T. Rilho and H. Madeira, "Joint Evaluation of Performance and Robustness of a COTS DBMS through Fault-Injection", in *Proc. Int. Conference on Dependable Systems and Networks (DSN-2000)*, New York, NY, USA, 2000, pp. 251-260 (IEEE CS Press).
- [20] H. Madeira, K. Kanoun, J. Arlat, Y. Crouzet, A. Johanson and R. Lindström, "Preliminary Dependability Benchmark Framework", DBench Project IST 2000-25425 Deliverable no. CF2, Available at <http://www.laas.fr/dbench/delivrables.html>, 2001.
- [21] K. Kanoun, M. Kaâniche and J.-C. Laprie, "Qualitative and Quantitative Reliability Assessment", *IEEE Software*, vol. 14, no. 2, pp. 77-86, mars 1997.
- [22] K. Kanoun and M. Borrel, "Dependability of Fault-Tolerant Systems - Explicit Modeling of the Interactions between Hardware and Software Components", in *IEEE International Computer Performance & Dependability Symposium (IPDS'96)*, Urbana-Champaign, IL, USA, 1996, pp. 252-261 (IEEE Computer Society Press).
- [23] J. Arlat, K. Kanoun, H. Madeira, J. V. Busquets, T. Jarboui, A. Johansson and R. Lindström, "State of the Art", DBench Project IST 2000-25425 Deliverable no. CF1, Available at <http://www.laas.fr/dbench/delivrables.html>, 2001.
- [24] W.-L. Kao, R. K. Iyer and D. Tang, "FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System Behavior under Faults", *IEEE Transactions on Software Engineering*, vol. 19, no. 11, pp. 1105-1118, November 1993.
- [25] K. Buchacker and V. Sieh, "Framework for Testing the Fault-Tolerance of Systems Including OS and Network Aspects", in *Proc. High-Assurance System Engineering Symp. (HASE-2001)*, Boca Raton, FL, USA, 2001, pp. 95-105.
- [26] T. Jarboui, J. Arlat, Y. Crouzet and K. Kanoun, "Experimental Analysis of the Errors Induced into Linux by Three Fault Injection Techniques", in *Proc. Int. Conf. on Dependable Systems and Networks (DSN-2002)*, Washington, DC, USA, 2002 (IEEE CS Press).
- [27] F. Raab, "Overview of the TPC Benchmark™ C: A Complex OLTP Benchmark", in *The Benchmark Handbook for Database and Transaction Processing Systems*, (J. Gray, Ed.) pp. 131-267, San Francisco, CA, USA: Morgan Kaufmann Publishers, 1993.
- [28] J. Durães, D. Costa and H. Madeira, "Accuracy of the Emulation of Software Faults by Machine-Code Level Errors", in *Supplement of the Int. Conference on Dependable Systems and Networks (DSN-2001)*, Göteborg, Sweden, 2001, pp. B.92-B.93 (Chalmers University of Technology, Göteborg, Sweden).
- [29] Saab Ericsson Space, "Microprocessor Thor, Product Information", 1993.
- [30] J. C. Baraza, J. Gracia, D. Gil and P. J. Gil, "A Prototype of a VHDL-based Fault Injection Tool", in *Proc. Int. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT'2000)*, pp. 396-404. Yamanashi, Japan, October 2000. (IEEE CS Press).
- [31] K. Kanoun, J. Arlat, D. J. G. Costa, M. Dal Cin, P. Gil, J.-C. Laprie, H. Madeira and N. Suri, "DBench - Dependability Benchmarking", in *Supplement of the 2001 Int. Conf. on Dependable Systems and Networks (DSN-2001)*, Göteborg, Sweden, 2001, pp. D.12-D.15 (Chalmers University, Göteborg, Sweden). See also <http://www.laas.fr/dbench/delivrables.html>