DBench

Dependability Benchmarking

IST-2000-25425

Preliminary Dependability Benchmark Framework

Report Version: Deliverable CF2

Report Preparation Date: 30 August 2001

Classification: Public Circulation

Contract Start Date: 1 January 2001

Duration: 36m

Project Co-ordinator: LAAS-CNRS (France)

Partners: Chalmers University of Technology (Sweden), Critical Software (Portugal), University of Coimbra (Portugal), Friedrich Alexander University, Erlangen-Nürnberg (Germany), LAAS-CNRS (France), Polytechnical University of Valencia (Spain).

Sponsor: Microsoft (UK)

Project funded by the European Community under the "Information Society Technology" Programme (1998-2002)

Table of Contents

Preliminary Dependability Benchmark Framework

Authored by: H. Madeira[++], K. Kanoun*, J. Arlat*, Y. Crouzet*, A. Johansson** and R. Lindström**

With the contribution of: S. Blanc[◆◆], K. Buchacker[◆], J. Durães[++], P. Gil[◆◆], T. Jarboui*, J.-C Laprie*, J. J. Serrano[◆◆], J. G. Silva[++], N. Suri** and M. Vieira[++]

* LAAS ** Chalmers [++] FCTUC [◆] FAU [◆◆] UPVLC

August 30, 2001

Abstract

The goal of dependability benchmarking is to provide generic ways of characterising the behaviour 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 uniform, repeatable and cost-effective way of performing this evaluation either as stand alone assessment or more often for comparative evaluation across systems. This document presents the preliminary framework proposed by the DBench project to investigate, define, and validate dependability benchmarks for computer systems, with particular emphasis on COTS-based systems and COTS components. The multiple dimensions of the problem are discussed and the framework is presented through a set of dependability benchmarking scenarios, ranging from pure experimental approaches to benchmarking scenarios where modelling and experimentation are tightly coupled. The main problems and research goals behind each benchmarking scenario are discussed and two concrete examples of preliminary dependability benchmarks already under investigation are presented.

1 Introduction

A key driver for computer systems has been the provision of performance and its consequent ability to provide for enhanced functionality. However, the performance related features of a system are meaningful only if the services can be provided in a dependable manner. As we become more and more dependent on computing systems, the impact of these systems failing also grows – in terms of nuisance value and in terms of the cost impact of lack of service delivery. Consequently, the provision of dependability becomes a key aspect for the system performance to be meaningful. Unfortunately, while there are relatively straightforward and well-established ways to evaluate and compare performance of different systems or components, the evaluation of dependability features is a considerably more intricate process, as in addition to the characterisation of the behaviour of the system under normal operational conditions, the system has also to be characterised under faulty conditions.

The use of Commercial Off-The-Shelf (COTS) components in a wide range of computer systems, including systems used in mission-critical and business-critical applications, is another growing trend. System designers perceive COTS as an opportunity to reduce development costs and deployment times. Additionally, COTS normally benefit from a large installation base in a multitude of configurations, which is often treated as effective field-testing. However, in spite of these advantages, COTS are usually not designed for stringent requirements of safety or service critical systems. Thus, it becomes important to have means to measure the impact of use of COTS in the computer dependability in general and particularly in business-critical and mission-critical systems.

It is well known that evaluation of dependability features is a complex task. First, the measures for dependability are inherently complex and dependent on a multitude of external aspects, such as the operational environment and particularly human aspects related to the maintenance and use of computer systems. Even the validation of very specific error handling mechanisms such as error detection and recovery mechanisms, which are relatively minor parts of the overall dependability evaluation, is traditionally a challenging task. Furthermore, the notion that functionality and performance are related to dependability is an established facet, as many systems operate often in degraded modes or with reduced performance due to faults or other unavoidable upsets. Operating in degraded modes is usually desirable, though it complicates even further the notion of defining and measuring dependability.

Given the huge complexity involved in the design of a computer system (hardware, operating system, application software, user interface, etc.), there is no single generalised approach for dependability evaluation and validation. Instead, several methods have been used, ranging from pure modelling and analytical techniques to simulation and experimental approaches based on fault injection and robustness testing [Kanoun *et al.* 1997, Folkesson & Karlsson 1998, Sinha & Suri 1999, Koopman & DeVale 1999, Costa *et al.* 2000]. Most of these techniques have been developed for mission-critical systems or for the high-end business-critical area, and thus make assumptions about design or operating environment that affect their direct porting to more mainstream computing systems.

The goal of benchmarking the dependability of systems is thus to provide generic ways of characterising the behaviour of components and computer systems in the presence of faults, allowing for the quantification of dependability measures. Beyond existing experimental techniques, such as fault injection and robustness testing, dependability benchmarking must provide a uniform, repeatable and cost-effective way of performing this evaluation either as stand alone assessment or more often for comparative evaluation across systems. In practice, a dependability benchmark must include a precise specification of the dependability measures to be taken, a detailed specification of all the procedures and steps required to obtain the benchmark measures (which may include programs source code, scripts, specification language texts, etc.), and the domain in which these measures are valid and meaningful.

Computer benchmarking is primarily based on experimental approaches. As an experiment, its acceptability is largely based on two salient facets of the experimental method: 1) the ability to reproduce the observations and the measurements, either on a deterministic or on a statistical basis, and 2) the capability of generalizing the results through some form of inductive reasoning (although the generalization is often limited by many factors). 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 on a statistical basis.[1] On the other hand, the necessary generalisation of the results is inherently related to the representativeness of the experiments. The notion of representativeness is manifold and touches almost all the aspects of benchmarking, as it really means that the measures and the conditions used to obtain those measures are representative of what can be found in the real world.

The key aspect that distinguishes benchmarking from existing evaluation and validation techniques is that a benchmark fundamentally represents an agreement (explicit or tacit) that is accepted by the computer industry and by the user community. This technical agreement (the benchmark) states the measures, the way the measures are obtained, and the domain (e.g., application area) in which these measures are valid and meaningful. In other words, a real benchmark is something that the user community and the computer industry accept as representative enough of a given application domain to be deemed useful and to be generally used as a way of measuring specific features of a computer system and, consequently, a way to compare different systems.

The concept of benchmarking can then be summarised in three words: *representativeness*, *usefulness*, and *agreement*. A benchmark must be as representative as possible of a given domain but, as an abstraction of that domain, it will always be an imperfect representation of reality. However, the objective is to find a useful representation that captures the essential elements of the given domain and provides practical ways to characterise the computer features that help the vendors/integrators to improve their products and help the users in their purchase decisions. In this context, the goal of the DBench project is to advance fundamental aspects

---

[1] Even in simple performance benchmarks, results such as peak performance are difficult to repeat exactly in successive benchmark experiments, and the benchmark result represents normally the best result achieved in the conditions specified by the benchmark procedures.

related to the representativeness and usefulness of dependability benchmarks, and to create the technical conditions for the establishment of actual dependability benchmarks that can be accepted and used by the computer industry and the user community.

Dependability benchmarking, as a specific form of benchmarking of computer systems, inherits the basic characteristics of benchmarking discussed above. Unfortunately, while classical benchmarking related to performance and functionality is relatively straightforward, the characterisation of dependability features is much more difficult. In fact, as mentioned before, the assessment of dependability attributes adds several layers of complexity to the problem. For example, the relevant measurements necessary to portrait system or component dependability are complex, and become even more complex when the operation in degraded modes or with reduced performance in the presence of faults is considered. The representativeness of the experimental steps required to assess dependability related measures is a very complex problem, and the need of using modelling techniques to evaluate basic dependability attributes such as availability and reliability is something completely new in the benchmarking context.

The success of well-established performance benchmarks in comparing performance of components and computer systems probably accounts for the generalised idea that the goal of benchmarks is to compare systems based on benchmark results. Although the comparison of dependability aspects of different alternatives is one of the goals of the dependability benchmarking effort, we would like to emphasise that the DBench project addresses many other goals for dependability benchmarks. Clearly, the key objective of dependability benchmarking is to contribute to improve computer system dependability. This central objective entails multiple perspectives for the use of future dependability benchmarks, ranging from end-user to the system developer perspective, and will be materialised in DBench through the definition of dependability benchmarks for the following specific objectives:

- ***Identify*** malfunctioning or less robust parts requiring more attention and perhaps needing improvements by tuning a particular component to enhance its dependability (e.g., by using wrapping), or tuning the system architecture (by adding fault tolerance mechanisms or spare units, for example) to ensure an appropriate dependability level.
- ***Assess*** the dependability of a component or a system.
- ***Compare***, grade or rank the dependability of alternative or competitive solutions.

Benchmarks are normally targeted to well defined application areas or to specific types of systems. This is a practical attitude to cope with the huge diversity of applications and systems in the computer industry. However, in spite of this inevitable diversity, we believe that different dependability benchmarks should share a common framework. This framework must define all the key components of a dependability benchmark and the general approach of benchmarking for dependability. In this context, an actual dependability benchmark is just an instantiation of this general framework to a specific application domain or to a particular kind of computer system or component.

The advantages of defining this preliminarily dependability benchmark framework at the beginning are manifold:

- Definition of all key dimensions that characterise different types of dependability benchmarks.
- Identification of the key components and the basic methods that characterise any dependability benchmark.
- Definition of dependability benchmarking concepts and common vocabulary to be used along the project.
- Identification of the points in the problem space that should be chosen as initial target of the dependability benchmarking research.
- Definition of additional experiments required to better understand the problem and to assess the technologies to be used in dependability benchmarking.

It is expected that the outputs of dependability benchmarks will be based on some dependability benchmark measures, evaluated under specific conditions. Therefore, an important task in benchmark definition concerns the identification of meaningful dependability benchmark measures. This is also the reason why we devoted an entire chapter (Chapter 2) to the presentation of dependability measures in a structured manner.

The structure of this document is as follows[2]: the next chapter presents the dependability measures. Chapter 3 presents the dimensions of the dependability benchmarking and discusses possible choices that define the boundaries of the problem in the context of DBench. Chapter 4 presents the framework through a set of dependability benchmarking scenarios and discuss the main problems and research goals behind each benchmarking scenario. Chapter 5 proposes preliminary dependability benchmarks examples already under investigation in DBench. Conclusions are presented in Chapter 6.

2 Measures of Dependability

The dependability characteristics of a system or a component can be expressed in a qualitative manner (in terms of attributes and specific features giving the system capacities and properties) or in terms of quantitative measures. The aim of this chapter is to present examples of attributes, features and measures that can be used to characterise the dependability of a system or a component.

As the occurrence or activation of faults in a system 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, in addition to dependability measures, will allow characterising completely the system behaviour from the dependability point of view. We will thus address performance measures under faulty conditions.

Dependability is an integrative concept that encompasses the following basic attributes [Laprie 1995, Avizienis, 2001]:

2 The state-of-the-art on dependability assessment, robustness and performance benchmarks is given in a separate deliverable, CF1 [CF1 2001].

- Availability: readiness for correct service.
- Reliability: continuity of correct service.
- Safety: absence of catastrophic consequences on the user(s) and the environment.
- Confidentiality: absence of unauthorised disclosure of information.
- Integrity: absence of improper system state alterations.
- Maintainability: ability to undergo repairs and modifications.

Several other dependability attributes have been defined that are either combinations or specialisations of the six basic attributes listed above. Security is the concurrent existence of a) availability for authorised users only, b) confidentiality, and c) integrity with 'improper' taken as meaning 'unauthorised'. Dependability with respect to erroneous inputs is referred to as robustness.

The attributes of dependability may be emphasised to a greater or lesser extent depending on the application: availability is always required, although to a varying degree, whereas reliability, safety and confidentiality may or may not be required.

The extent to which a system possesses the attributes of dependability should be interpreted in a relative, probabilistic sense, and not in an absolute, deterministic sense: due to the unavoidable presence or occurrence of faults, systems are never totally available, reliable, safe, or secure.

The evaluation of these attributes leads to view them as measures of dependability. The associated measures are referred to as **comprehensive dependability measures** as i) they characterise the service delivered by the target system, ii) they take into account all events impacting its behaviour and their consequences and iii) they address the system in a global manner, even though the notion of system and component is recursive and a system may be a component of another system.

It is worth noting that all measures given in this chapter may not be relevant for all systems and users, but they can all be found on many lists of user concerns and establish the general premise of benchmarks.

In the following sections we first provide examples of comprehensive measures, then examine specific system features together with their associated specific measures and finally we address related performance measures.

2.1 Comprehensive Dependability Measures

Comprehensive measures characterise a system or a component in a global manner, addressing the service delivery level and taking into account the occurrence of the various events impacting its behaviour.

Definitions of reliability, availability, maintainability and safety as measures of dependability are given in Table 2.1 [Laprie 1995, Avizienis, 2001].

Reliability measures the continuous delivery of correct service or, equivalently, the time to failure.
Availability measures the delivery of correct service with respect to the alternation of correct and incorrect service.
Maintainability measures the time to service restoration since the last failure occurrence, or equivalently, measures the continuous delivery of incorrect service.
Safety: when the state of correct service and the states of incorrect service due to non-catastrophic failure are grouped into a safe state (in the sense of being free from catastrophic damage, not from danger). Safety measures the continuous safeness, or equivalently, the time to catastrophic failure. As a measure, safety is thus reliability with respect to catastrophic failures.

Table 2.1: Comprehensive dependability measures

The above measures could be evaluated from direct observation of system behaviour. However, a long observation period or a large number of systems are needed to obtain statistically significant numbers related to measures such as system availability or mean time to failure. Usually, comprehensive measures are obtained based on modelling.

Indeed, dependability modelling based on Markov chains and Petri nets has been established as an efficient methodology for evaluating the comprehensive dependability measures, even before system deployment. Dependability modelling requires the knowledge of the following:

- System functions and structure.
- System nominal and degraded operational modes, system failure modes, as well as component failure modes.
- The most significant error detection and fault-tolerance capabilities.
- Maintenance facilities (e. g., on-line repair and backup possibilities, maintenance policy and maintenance duration).
- System evolution capacities (e.g., possibility of on-line upgradability and detection of inconsistent upgrades).

Information related to the three first points are obtained from system analysis with the support of the system constructor.

Information related to maintenance can be obtained from the system constructor and from the system owner (as the knowledge of the environment of system utilisation is usually required).

Information related to system evolution could be obtained from system constructor and from experimentation.

For modelling purposes, the above sets of information are expressed in terms of event occurrence rates and conditional probabilities. Events refer for example to fault occurrence and

activation, error propagation and system repair or restart. Conditional probabilities are usually related to system behaviour after event occurrence. For example, if the event is fault occurrence and activation, the conditional probabilities describe the reaction of the system to faults (i.e., probability of error detection, probability of error containment or probability of error recovery).

Problems related to model construction and processing are discussed in the companion state-of-the art document [CF1 2001] and will not be examined further in this document. It is worth noting that some assumptions have to be made to construct manageable models. Sensitivity analysis with respect to these assumptions is essential to check their validity as well as their impact on the results.

The evaluation of dependability measures based on the constructed system model entails additionally the knowledge of the numerical values of the model parameters (event rates and conditional probabilities). From a practical point of view:

- Fault occurrence and activation rates are generally obtained from field measurement related to the same system if it is already in operation or, most probably, from similar previous systems that have already been in operation for a long time.
- Conditional probabilities can be obtained from experimentation: either from field data or from controlled experiments, whenever possible.

When the parameter values are not available, approximate values can be attributed in a first step. Sensitivity analyses allow identification of the most salient ones for the considered measure(s), to be evaluated from experimentation.

2.2 Dependability Features

Attributes and comprehensive measures allow the characterisation of a system or a component in a global manner (at the service delivery level, taking into account all events impacting its behaviour and their consequences). However, for several reasons, it may be interesting to characterise only specific aspects of system behaviour without necessarily taking into account all the processes impacting its global behaviour and without addressing the service delivery level (i.e., characterise the system fault tolerance capabilities or its maintenance and evolution capacities individually). This is done through the study of what we refer to as dependability features.

Without being exhaustive, Table 2.2 illustrates the kind of features that can be considered individually. Indeed, a particular system may have a subset of the features given in Table 2.2 as well as other features that are considered to be important for that particular system. For example, features such as extendibility, scalability and modularity may be considered as essential for a system, even though they are not directly related to dependability, but they may impact system dependability.

Error detection, error containment and system recovery: - Detection / recovery of permanent hardware and/or software faults - Detection / recovery of transient hardware and/or software faults - Detection / recovery of successive faults - Error containment (avoidance of error propagation) - On-line fault diagnosis - Protection against operational errors (accidental / intentional) - Failure modes - Recovery after power failure
Maintenance and evolution: - On-line repair - On-line backup - On-line upgradability (new release) - Detection of inconsistent upgrade

Table 2.2: Example of dependability features

The list of features is provided in a generic manner, and each feature has to be specified precisely to characterise the dependability of a system. For instance, one has to specify the exact nature and location of errors that are detected, contained (whose effects are confined) or tolerated. For example, a system may be tolerant to hardware faults without being tolerant to software faults. The failure modes should be specified and defined clearly in the context of the particular application. In particular, for some systems, correct output values delivered too late with respect to system specifications are considered as leading to a system failure. Moreover, some systems can be designed and implemented so that they fail only in specific and controlled modes of failure. Such systems are fail-controlled systems. When failures are halting failures only, to an acceptable extent, the system is considered as a fail-halt or a fail-silent system; when failures are all minor ones, to an acceptable extent, the system is considered as a fail-safe system.

Features are suitable for describing qualitatively the dependability of a system. However, in order to have an accurate knowledge about the system behaviour, features should be completed by quantitative information. In particular, one has to know to which extent these features are fulfilled. This leads to associate to each feature one or more **specific dependability measures** that should be quantified to better describe it.

2.3 Specific Dependability Measures

Table 2.3 illustrates examples of specific measures that can be associated with the features listed in Table 2.2. These measures should be defined accurately in order to be useful. As happen with the features, one has to precise the nature of errors that are detected, contained or covered.

As can be seen, most of the features are characterised by their efficiency. Efficiency has two complementary dimensions: i) a time dimension corresponding to the duration of the considered action (error detection, recovery or containment, fault diagnosis and system repair) and ii) a conditional probability of success of an action provided it has been initiated (also referred to as coverage or coverage factor). For example fault diagnosis coverage is defined as the probability that a fault is correctly diagnosed given the fact that an error is detected. However, for some systems, action duration or action coverage may have more impact and emphasis may thus be put only on the most influential dimension of efficiency.

Error detection, error containment and system recovery: - Error detection efficiency - Error recovery efficiency - Probability of detection/recovery of successive faults - Error containment efficiency - Fault diagnosing efficiency - Efficiency of detection/recovery of operational errors - Failure modes (quantification of %) - Efficiency of recovery after power failure
Maintenance and evolution - Repair efficiency - Time to restart the system - Time to system back-up - Time to upgrade, probability of detection of inconsistent upgrade

Table 2.3: Example of specific system dependability measures

The variety and number of measures show the complexity of dependability characterisation. Each measure characterises one side of the multi-faceted problem. However, the system can be globally characterised by combining the most salient specific measures into a global model to evaluate appropriate comprehensive dependability measures as in Section 2.1. The various measures are represented by parameters in the model. A further step would be to associate numerical values to these measures/parameters.

2.4 Performance Measures

Classical performance measures include measures such as system response time and system throughput. These measures are evaluated based on analytical models, simulation models or measurements. As for dependability measures, measurements are possible only if the system under study does exit while analytical modelling and simulation can be used for situations where measurement is not possible.

Table 2.4 gives the definition of response time and throughput as defined in [Jain, 1991]. In computer systems shared by many users, the response time and throughput may be measured for each user as well as for the whole set of users.

Response time is defined as the interval between a user's request and the system response, assuming that the request and the response are instantaneous. Taking into account the request time and time for outputting the response leads to two different definitions: - The response time is the interval between the end of a request and the beginning of the corresponding response from the system. - The response time is the interval between the end of a request and the end of the corresponding response from the system. **Throughput** is defined as the rate (requests per unit of time) at which the requests can be serviced by the system. Some examples: For CPUs, the throughput is measured in Millions of Instructions Per Second (MIPS) or Millions of Floating-Point Operations Per Seconds (MFLOPS). For transactional processing systems, the throughput is measured in Transactions Per Second (TPS).

Table 2.4: Performance measures

Indeed, the performance evaluation allows the characterisation of system behaviour with respect to the additional fault tolerance mechanisms and in presence of faults. For example, some fault tolerance mechanisms may have a very high coverage factor with a large time overhead in normal operation. It is interesting to evaluate such time overhead. Concerning the system behaviour in presence of faults, following fault occurrence or fault activation, either the system fails or a correct response is provided (correct value, delivered on time). Indeed, a correct value delivered too late with respect to the system specification is to be considered as a failure. This can be considered as a specific failure mode.

In case of a correct response, the system performance may be affected by the presence of errors in the system. The evaluation of the response time and throughput measures allows the characterisation the **system performance in presence of faults**. The performance measure definitions given in Table 2.4 still hold, but it is assumed that these measures are obtained in presence of faults.

2.5 Comments on Measures of Dependability

Previous sections presented the various dependability attributes and features and their associated measures that allows characterisation of the dependability a system or a component as well as performance measures. We have distinguished two kinds of dependability measures: comprehensive and specific measures. Comprehensive measures characterise the system globally at the service delivery level, taking into account all events impacting its behaviour and

their consequences. Specific measures characterise particular aspects of a system or a component: behaviour in presence of fault, maintenance and evolution. Comprehensive measures are usually evaluated based on analytical modelling. Some of the specific measures may appear as parameters in the analytical model.

It is worth mentioning that as the notion of system is recursive (as stated in the introduction of this chapter), the concepts of comprehensive and specific features are applicable at the system and component levels. However, when considering software components, COTS or not, emphasis is usually placed on specific dependability measures rather than on comprehensive dependability measures. The most salient dependability features are those related to error detection, containment and recovery, and failure modes. As an example Table 2.5 shows the dependability features of an operating system. Also, robustness is a commonly used feature for characterising an operating system. Robustness characterises the operating system reaction with respect to erroneous inputs (e.g., illegal Operating System calls).

Error detection, containment and recovery:
- Detection / recovery of hardware faults
- Detection / recovery of middleware / application software faults
- Detection / recovery of its one faults
- Error containment (or error propagation channels)
- Failure modes
- Recovery after power failure

Table 2.5: Example of an operating system dependability features

When considering the system level, we have implicitly assumed a locally distributed system, with well-defined boundaries and a limited number of dependability measures. When considering largely distributed systems of systems as for web-based applications, the definition of dependability measures may be more complex as well as the analytical modelling of such systems. However, a hierarchical analytical modelling approach can be used as in [Kaâniche, 2001] to master the model complexity. The number of parameters may be larger as specific features and measures can be considered (see e.g., [Labovitz *et al.* 1999]), but the principles described in this section are still applicable.

Finally, measures obtained experimentally require measurements on the target system. Hence the need for the definition of an appropriate workload and possibly an appropriate set of faults to be injected in the system, referred to as faultload in the rest of this report. The selection of the right workload and faultload is mandatory to obtain significant results.

3 Dimensions and Boundaries of the Dependability Benchmarking Problem Space

Dependability benchmarking is an n-dimensional problem space. The definition of a framework for dependability benchmarking requires first of all the identification and clear understanding of all the dimensions of the problem. Then, defining the framework corresponds to making

choices for the different dimensions and characterising all the key components of dependability benchmarks. In this sense, different frameworks for different flavours of dependability benchmarking can co-exist and the options assumed in the preliminary framework represent just the starting point of the project research. Refinements in the framework (or even alternative dependability benchmark frameworks) may be considered during the course of the project.

Although benchmarking the process of creating[3] a computer system or component could be conceivable (in fact, the ISO 9000 series of standards are based on the idea of certifying processes), the DBench approach is to benchmark actual products via experimentation and modelling. The idea of benchmarking a product versus benchmarking the process of creating a product is a key first dimension of the problem, and the focus on product benchmarking represents a key decision for DBench. By benchmarking actual computer products we will bring the end-user perspective to the benchmark results, while we also encompass the indirect evaluation of product development phases, through the characterisation of the developed product. In addition to the end-user, who is mainly interested in dependability benchmarking focusing the operational phase of the computer systems (which includes the actual use of the system, maintenance actions, and system evolution and upgrading), benchmarking is also very relevant for the system developers and integrators, as it provides them with means to select the best components or the best system configurations.

As stated before, a dependability benchmark can be represented by a well-defined set of dependability measures and a detailed specification of all the procedures, methods, tools, and steps required to obtain these measures. In this sense, the measure dimension is the most important one, as it defines what is expected from a dependability benchmark. However, in order to understand the problem space and to define the boundaries of the research work in DBench we also have to take into account all the other dimensions of dependability benchmarking. To facilitate this discussion we organise the presentation of the dimensions in four groups, namely categorisation dimensions, measure dimension, experiment dimensions and property dimensions.

- **Categorisation dimensions** – These dimensions characterise the dependability benchmarks and naturally define a set of different benchmark categories. In practice, we have as many dependability benchmark categories as possible combinations of the categorisation dimensions (although only some combinations are actually useful). The presentation of these dimensions in first place allows us to focus the discussion of the other dimensions, as these dimensions define in practice the general bounds of the problem. Obviously, the product versus the process benchmarking is the very first categorisation dimension, but given the fact that DBench only considers product benchmarking, it is not necessary to include this dimension in the discussion anymore. Thus, the categorisation dimensions include:
  - *Benchmark usage* – identifies the different perspectives for running dependability benchmarks and using the benchmark results. This is in fact a composite dimension

3 We use "create" to designate all the complex chain of processes (from specification to product deployment) that result in an actual computer system or component.

including sub-dimensions such as benchmark performer, benchmark user, results use, and results scope (see Section 3.1.1).

- *Life cycle phase* - identifies the phase in the product life that will be addressed in the benchmark.
- *Application area* – identifies the application area defined with the adequate granularity (specificity).
- *Target system* - defines the target system and/or the target component that will be subject to measures.

- **Measure dimension** – The selection of dependability benchmarking measure(s) to be assessed depends on the choices made for the categorisation dimensions. The measures of dependability have already been presented in Chapter 2 and, in this section, we discuss particular aspects induced by the benchmarking context.
- **Experiment dimensions** – Once defined the main sets of measures that can be obtained from a dependability benchmark, we discuss the dimensions related to the experiments on the target system required to get the measures and the base data needed to compute the dependability measures (this computation can range from simple calculations to complex modelling). The experiment dimensions are:
  - *Operating environment* - typical environment for an application area and the way it affects the benchmark.
  - *Workload* - defines a working profile that should be representative of an application area.
  - *Faultload* - defines the set of upsets, stressful conditions and faults that could affect the system.
  - *Procedures and rules* - defines the procedures and the rules to perform the benchmarking (this includes more than just the aspects directly related to the experimental steps of benchmarking).
- **Property dimensions** – The dimensions mentioned above include all the aspects needed to define a dependability benchmark. However, dependability benchmarks must meet some properties to be valid and useful. For example, a benchmark must be repeatable (in statistical terms), must not cause undesirable interferences with the target system, must be portable, cost effective, etc. In practice, all these aspects constitute another set of dimensions of the problem. However, due to the large number of different properties and to the fact that dependability benchmark properties are dependent on specific benchmark components, they will be discussed in more detail in Section 4.2.

In the remaining of this chapter we present and discuss these groups of dimensions of dependability benchmarking. This general discussion is essential to understand the problem space and to define the boundaries of that space from the DBench point of view. For this reason, after presenting each dimension in a neutral perspective, we will discuss the different possibilities from the DBench point of view, assuming choices that clarify our vision of the

problem. This focusing approach is also essential to reduce the complexity of the problem and make it tractable, while keeping all the relevant and innovative aspects of dependability benchmarking.

3.1 Categorisation Dimensions

3.1.1 Benchmark Usage

Performing a dependability benchmark and using the benchmark results can be done in multiple perspectives. The benchmark usage is in fact a composite dimension including the following sub-dimensions:

- Benchmark performer – Person or entity who performs the benchmark and is responsible for the compliance (concerning the benchmark specification) of the benchmark results. Some examples are system/component developer and vendor, system integrator, end-user, third party such as independent organisations, computer magazines, etc.
- Benchmark user – Person or entity that actually uses the benchmark results, who could be different from the benchmark performer.
- Results use – Benchmark results can be used for different purposes. For example, to characterise system dependability capabilities, to assess the efficiency of these capabilities, to identify weak points, to compare alternative systems or components, etc.
- Results scope – Benchmark results can be used internally or externally. Internal use means that the benchmark is used as a tool to help in achieving specific goals of the benchmark user (person or entity). For example, helping the vendor to improve their products or helping the end-users to tune their systems. External use means that benchmark results are standard results that fully comply with the benchmark specification (this is the case of many performance benchmark results).

The different usage perspectives affect the type (and the detail) of measures expected from the benchmark and the type of target system. Typically, end-users are interested in general system dependability measures (e.g., unconditional system availability) while a system integrator or a developer could be more interested in specific measures related to the behaviour of the system or a component in the presence of faults (e.g., error detection coverage or recovery efficiency). Thus, the different user perspectives potentially create demands for different types of measures from the dependability benchmarks.

DBench addresses all the different usage perspectives mentioned above, which can be combined and crossed in several ways. However, the main goal of DBench is to move dependability benchmarks from the *ad hoc* methods that characterise internal use to standard approaches required for the external use of benchmark results.

3.1.2 Life cycle Phase

The main phases of a typical life cycle of a computer system or component are the requirements, design, implementation/manufacturing, deployment, and operational phases. A

dependability benchmark could be specifically targeted to any of these phases (e.g., design: evaluate design correctness or design robustness to component failure; implementation/manufacturing: evaluate the impact on dependability of manufacturing defects or coding errors, etc.). However, a product-based benchmarking (as desired for DBench) means that we are particularly interested in benchmarking dependability features at the operational phase of the target system or component, which includes normal system operation and maintenance in a realistic environment. System maintenance is particularly relevant and one important goal of DBench is to devise ways to take maintenance into account in the dependability benchmarks.

Other perspectives mentioned in Section 3.1.1 for the use of benchmark results address other life cycle phases. For example, the identification of weak points in a system and consequent improvement of the system is normally related to phases of product development such as testing and validation. These life cycle phases are also considered in DBench.

3.1.3 Application Area

The application area is a key dimension. The division of the application spectrum into well-defined application areas is necessary to cope with the diversity of systems and applications, and to make it possible to make choices on most of the other dimensions. In fact, most of the dimensions and dependability benchmark components are very dependent on the application area. For example, the benchmark measures, the operational environment, or the most common (external) faults that may affect the systems and the workload (just to name a few of them) are very dependent on the application area.

The main difficulty with this dimension is clearly the establishment of the right granularity to divide the application spectrum. The application areas must be as general as possible but, at the same time, specific enough to allow the definition of all the aspects directly dependent on the application area. Obviously, different application areas will tend to need different dependability benchmarks.

The research activities in DBench will consider two application areas: transactional and embedded applications. This is clearly a coarse grain division and one of the tasks required is the refinement of this classification. The simple observation of the way performance benchmarks handle the problem of the application area can give us some insights on the grain required to divide the application spectrum. For example, the Transaction Processing Performance Council (TPC) proposed four different benchmarks for (database oriented) transaction applications: the TPC-C for traditional operational database applications, TPC-W for e-commerce applications, and the TPC-R and TPC-H for decision support applications, being the first one for report oriented applications and the second one for *ad hoc* decision support queries (see CF1, Chapter 5 for a more details on TPC benchmarks [CF1 2001]).

These insights from other benchmarks are merely indicative and the actual criteria for defining application areas for dependability benchmarking is clearly one of the research goals of DBench. Nevertheless, the principle is basically the same for all kinds of benchmarks: *the application area must be specific enough to make the definition and agreement on all the components of a dependability benchmark possible.*

3.1.4 Target System

The definition of the target system and the identification of the adequate level of detail required for its description in a dependability benchmark are very important problems for DBench. Two contradictory forces determine the way the target system should be described in a dependability benchmark. On one hand, it would be desirable to know the target system in detail to make possible the evaluation of quite specific measures such as dependability measures directly related to fault tolerance mechanisms available in the system. On the other hand, a detailed description of the target system (i.e., hardware and software architecture as well as implementation details) is seldom available and, if available, its inclusion in the dependability benchmark will drastically reduce the portability of the benchmark. In the limit, a very detailed description of the target system would make the benchmark applicable to only one system, which is exactly the opposite of the concept of benchmarking.

This problem will be investigated in DBench through several complementary approaches. Following are two examples:

- **Functional view** – In this approach the target system is described as being able to run the benchmark workload (i.e., the adequate workloads chosen to represent a given application area). That is, the features and functionalities required by the workload define the capabilities of the target system in an implicit way, including the dependability features. For example, the workload may require the target system to have a given amount of mass-memory, a minimum amount of main memory, capability of executing sets of operations in an atomic way (transactions), capabilities to recover from specified sets of faults, capabilities to force integrity constraints on the application data, etc. The only requirement should be that the system is able to run the workload as specified in the benchmark and no particular assumptions on the target system architecture or on the existence/implementation of specific techniques in the system are needed.

  It is worth noting that this functional view of the target system does reflect the actual situation in the field, as systems using different architectures and techniques are used to run similar applications. Furthermore, the choice of COTS and COTS-based systems as the main focus for the dependability benchmarking research in DBench suggests that the benchmarks being developed should not rely on detailed information on the architecture and implementation of the target systems. The main question that has to be investigated is to what extent this approach is going to affect the accuracy and the different types of dependability measures that can be obtained from the benchmarks. One possible way out could be to associate to some dependability measures a disclosing requirement, which specifies the detailed knowledge of the target system internals required to obtain that specific measure.

- **Abstraction layers view** – In this approach dependability benchmarks assume a given architecture for the target system and require the description of target system details to a given level of abstraction. The target system architecture can be described in a generic way as a set of abstraction layers (or components) that perform specific functions in the target system. In addition to the generic layered description of computer systems (e.g., hardware, operating system, middleware, applications), which can be more or less detailed (e.g., the operating system is composed by a micro kernel, drivers, etc.), we will be particularly

interested in the description of the key components for dependability benchmarking. This way, a dependability benchmark may explicitly assume that the target system has error detection components, fault diagnosis, system reconfiguration, error recovery, etc. Although specific implementation details on these components should not be considered, the benchmark may assume the knowledge of specific techniques that are used (or not) in the target system. For example, the error detection can be based on structural redundancy and voting or it can be just a set of behavioural checks. The main question that will be investigated in DBench concerning this approach is how to conciliate the need of describing the target system to a given detail with the necessary level of portability required for a benchmark.

The benchmarking of specific components shares the same basic problem concerning the more or less detailed view of the component that may be required by the dependability benchmark. Additionally, the benchmarking of a given component can be considered with the component in a synthetic environment or with the target component integrated in a well-documented system running the normal workload (i.e., a benchmark workload defined to benchmark the entire system and not only the target component).

3.2 Measure Dimension

The dependability measures presented in Chapter 2 already introduced the framework for the definition of dependability benchmark measures. Two main groups of dependability measures have been identified: comprehensive measures (such as reliability, availability, maintainability, and safety) and specific measures (such as error detection efficiency, error recovery efficiency and error containment efficiency). Also, two types of performance measures have been presented, related respectively to i) the baseline performance (i.e., performance in absence of faults) and to ii) the performance in presence of faults, to characterise the performance degradation due to fault activation when compared to the baseline performance.

The selection of the appropriate measure(s) to be evaluated is directly related to the benchmark category that will be performed, which is in turn directly related to the various categorisation dimensions presented in Section 3.1 (i.e., the benchmark usage, the life-cycle phase, the application area, and the target system nature). DBench will identify the most significant measures for a set of well-identified benchmark categories. Also, we will put emphasis on benchmarking dependability measures **experimentally**.

With respect to the experimental evaluation of the defined measures, it is expected that **specific dependability measures** can be evaluated experimentally following a well-defined benchmarking process. One of the aims of DBench is to define the experimental framework for evaluating this kind of measures. Concrete examples of specific dependability measures will be defined for each benchmark prototype planned for the Dbench.

The main problem for evaluating comprehensive dependability measures experimentally concerns the duration of the experiments, as the system behaviour is conditioned by fault occurrence and activation intervals that are usually very large compared to the expected experiment duration. The usual situation is that these measures are evaluated based on modelling. One of the goals of DBench is to define dependability benchmarks in such a way that the knowledge/information required for dependability modelling can be provided (or at

least facilitated) by the benchmark implementation and by the benchmark results obtained directly from the experimental steps of performing the benchmark. However, some experimental measures related to comprehensive measures may be obtained directly from benchmark experiments to characterise the target system or component in the context of the experimental environment defined in the benchmark. These measures can be quite useful to help improve systems/components or to compare different systems/components using a **standard experimental environment**. Some examples of these measures are:

- Target system availability during the benchmark running. Assuming that the target system is exposed to a given number of faults and stressful conditions, this measure provides a direct assessment of the target system availability during the experiment. It is given by (time during which the system is available / Experiment time). Obviously, this measure is not equivalent to the target system availability, but it can be quite useful to improve the target system configuration, to tune it for enhanced availability, or to compare different target systems with respect to expected availability (obviously, these comparisons must be done with great care as they ignore the real fault rates).

- Target system reliability during the benchmark running. This measure is the equivalent of the previous measure when applied to the reliability concept. It could be expressed in terms of the number of failures observed in the target system during the experiment.

- Number of integrity errors detected in the key data or computer results. This measures the impact of the faults and stressful conditions specified in the benchmark on the data integrity.

In the same manner as for dependability measures, experimental performance measures can be evaluated using a standard experimental environment. They concern for example:

- The baseline amount of work done by the target system during the benchmark running (or the rate in the form of work done per unit of time). Note that an experimental performance measure has real meaning under the conditions defined for the experiment and cannot be used outside that context (e.g., it does not make sense to compare performance measures from different performance benchmarks).

- The amount of work done by the target system in the presence of faults (e.g., number of transactions executed according to some transaction profile, etc). It measures the impact of faults and stressful conditions (specified in the benchmark specification) on the performance and favours systems with higher capability of tolerating faults, fast recovery time, etc.

The performance measures in both normal conditions (baseline) and in the presence of faults allow the measurement of degradation conditions, which includes reduced performance modes and/or operation in reduced functionality. It is worth noting, however, that the need for establishing baseline performance measures must not be confused with performance benchmarking, as in our case we just need a baseline performance (i.e., non optimised beyond normal tuning) to evaluate the relative effect of the faults and stressful conditions on the performance degradation.

To sum up, specific measures can be assessed directly from experimentation while comprehensive measures can either be approached through experimental measures obtained under standard experimental conditions or evaluated based on experimentation and modelling. The first two kinds of benchmark measures will be referred to as **experimental measures** (including comprehensive and specific measures). Finally, it is worth noting that the comprehensive dependability measures, when evaluated based on modelling, represent something new in the context of benchmarking, as existing performance benchmarks only rely on direct measures taken at the time of a specific benchmarking event.

A relevant aspect is that the experimental measures represent observation points in the target system. One important goal is to avoid (or minimise as much as possible) the interference with the target system that may result from collecting the direct measures (i.e., measurements). The investigation of techniques and instrumentation means for measurements with no (or with acceptable) intrusiveness is another important research goal of DBench.

3.3 Experiment Dimensions

3.3.1 Operating Environment

The operating environment is traditionally one of the things that affect system dependability. In fact, many computer faults are induced by external sources, and these are intimately related to the operating environment. The main problem is to identify a way to take into account operating environment features in the benchmarking process. This could be particularly difficult if we include the human aspects (operators and users) in the definition of operating environment.

One possible solution could result from the observation that the operating environment is very dependent on the application area. Thus, if the application area is well defined and bounded it should be possible to define a typical operating environment (to be used in experiments) for a given application area, which includes the set of faults that are agreed upon as typical for that application area. In other words, the operating environment is dependent on the application area and is one of the things to take into account in the definition of the faultload.

Also, the operating environment may affect the workload in a way that is normally ignored in performance workloads that concentrate essentially on functional aspects. Some of the aspects such as preventive and corrective maintenance actions can be actually dependent on the operating environment and not on the application area.

The starting hypothesis that will be investigated in DBench is to try to characterise a typical operating environment for quite specific application areas and devise ways to translate the key elements of the operating environment into the faultload and possibly the workload.

3.3.2 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 preventive and corrective maintenance actions that are conducted as part of routine operations for a given application area. For example, in a database server events such as system backups or log files management are included in the typical profile and should be considered in the workload. These are clearly key aspects for dependability benchmarks and normally they are ignored in the performance benchmark workloads. DBench will analyse and use available workloads from performance benchmarks as a first step, but great care will be taken to include specific aspects of the application profile that can influence the dependability measures.

3.3.3 Faultload

The faultload consists of a set of faults and stressful conditions that are intended to emulate the real exceptional situations the system would experience in the field. This is clearly dependent on the operating environment for the external faults, which in turn depends on the application area. Internal faults (e.g., software faults and some hardware 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.

The basic reasoning behind the statistical equivalence of faults lies in the fact that the same general classes of faults can be observed in different systems. In fact, all computers may be the subject of hardware transient faults, well-known classes of software faults, operation faults, etc. Although the fault effects are very different from system to system, the characterisation of each class of faults is relatively similar for different systems. For example, hardware transient faults share the same basic characterisation in all computers (e.g., they cause single bit flips or burst bit errors, the probability of occurrence depends on factors such as chip areas, transistor sizes, voltage thresholds, timing margins, etc.). In a similar way, the same basic classes of software faults can be observed in quite different systems, in spite of significant differences at the software architecture, programming language, used compilers, etc.

The definition of the faultload will inevitably be a pragmatic process based on observation, knowledge, and reasoning. Inputs from many different areas are needed to accomplish this task with success. Some examples are information from faults in the field, characteristics of the operating environment, specific features of the hardware and software technology, analysis of the development processes, or even information from experimental and simulation studies.

Four basic steps seem necessary to define representative faultloads:

- Definition and characterisation of all the classes of faults and stressful conditions that must be considered.
- Assignment of relative percentages to the different classes. This is clearly very dependent on the concrete application area and the target system.
- Find methods and techniques to emulate each class of faults.

- Define the mechanisms and instrumentation required to introduce this faultload in the target system in a way that can be acceptable for a benchmark (i.e., cost-effective, easy to use, non-intrusive, etc.).

The definition of representative faultloads is probably the most difficult part of defining a dependability benchmark and will be thoroughly investigated in DBench, particularly in the context of WP2. To make this problem tractable we will start by focusing well-defined application areas in the context of the dependability benchmark prototypes planned in the project. Indeed, the behaviour of the system results from the combined effects of the workload and the faultload and it is important to define the fauldload knowing the workload of the system.

3.3.4 Procedures and Rules

A dependability benchmark must include standards for conducting experiments and to ensure uniform conditions for measurement. These standards and rules must guide all the processes of producing dependability measures using a dependability benchmark.

One aspect that has great impact on these procedures and rules is the final form of the “product” dependability benchmark. As it is not plausible to achieve dependability benchmarks in the form of something ready to use (as some of the simplest performance benchmarks that basically consist of running a program) the expected shape of dependability benchmarks is a detailed specification (obviously, tools and code meant to facilitate the benchmark implementation can go together with the benchmark specification). In this sense, the procedures and rules must start by defining the way the specification should be implemented in the target system. Additionally, the following aspects must be included in the procedures and rules:

- Standardised procedures for applying the benchmark. This includes uniform conditions for set up, initialisation, running and reading the direct benchmark results and all the information related to the production of the final measures from the direct results such as calculation formulas, ways to deal with uncertainties, errors and confidence intervals for possible statistical measures, etc.

- Scaling rules to adapt the same benchmark to systems of very different sizes (but used in the same application area). These scaling rules would define the way the system load can be changed. At first sight, the scaling rules are related to the baseline performance measures and should mainly affect the workload, but one task of DBench will be to investigate the need of scaling up or down other components of the dependability benchmark to make it possible to use the same benchmark in systems of quite different sizes.

- System configuration disclosures for performance measures. Dependability measures might also include requirements or disclosures involving all factors that affect dependability.

- Rules to avoid "gaming" to produce optimistic biased results.

3.4 Summary

The discussion of the dependability benchmark dimensions in previous sections pointed out several preferences and research directions that actually shape the preliminary dependability benchmark framework. The following points summarise these choices:

- Focus on the benchmarking of products through experimentation and modelling.
- Multiple benchmark usage perspectives are considered, including different kinds of benchmark performers and users (e.g., end-user, system integrator, system developer, etc.), and different uses for the benchmark results. In particular, one goal of DBench is to extend the internal use of dependability benchmarks (e.g., to characterise and improve the dependability of systems and components) into more standardised approaches required for the external use of benchmark results (e.g., to asses and compare alternative solutions). Normally, different usage perspectives mean different benchmark categories.
- Products are normally benchmarked as they are used in their operational phase, which includes both normal operation and maintenance actions. Dependability benchmarking of products in specific phases of the product development is also considered.
- Different approaches considering the level of detail used to describe the target system and target component in the dependability benchmarks will be considered. Examples of alternatives under research range from a simple functional view to the architectural description of the target according to an abstract model previously defined in the benchmark.
- Benchmarking of specific components can be done considering the component in a synthetic environment or with the target component integrated in a well-documented system running the normal workload.
- A wide range of different measures is required, including comprehensive and specific dependability measures, as well as performance related measures. The actual measures used in future benchmarks will be typically a subset of these measures.
- The application area is a key dimension and determines in practice many other dimension choices. This favours the end-user perspective as the dependability benchmark framework will be oriented towards benchmarking systems when doing typical tasks (from end-user point of view) of a given application area.
- An application area can be represented by a workload (from the dependability benchmark point of view).
- Typical operating environment can be characterised for each application area (assuming that the application spectrum is divided into well-defined application areas).
- The set of faults and stressful conditions can be defined from the characterisation of the operating environment and from general well-known classes of faults.
- Equivalent sets of faults for different target systems must be understood in a statistical basis.

- The product "dependability benchmark" could be a specification and a list of tools, methods, and techniques required to run the benchmark. In other words, it could be a document specifying the dependability measures, the way these measures are obtained, and the domain in which these measures are valid and meaningful.

4. Dependability Benchmarking Overview

This chapter presents a first overview on dependability benchmarking. Given the multifaceted nature of the problem, the different approaches that are being followed in the DBench project are presented through different examples of dependability benchmarking scenarios, starting from the simplest one and ending in the most general scenario. The main problems and research goals behind each scenario are discussed. These scenarios are related to the experiments achieved for benchmarking a system. However, before performing the experimental work itself, some analysis and preparations are required to select the appropriate experiments. We will thus present the various steps that have to be followed for conducting a dependability benchmark, before presenting the benchmark scenarios. The different benchmark properties required to attain useful and relevant benchmarks are surveyed in Section 4.2.

4.1 Dependability Benchmark Conducting and Key Benchmark Components

The selection of the appropriate experiments to carry out on the target system is strongly related to the category of benchmark that is considered and the features or measures considered in the benchmark. In turn, the benchmark category is itself related to the various categorisation dimensions presented in Chapter 3 (i.e., the benchmark usage, the life cycle phase, the application area and the target system nature). Different possibilities for these categorisation dimensions result in several categories of dependability benchmark. As a consequence, the first step in conducting a benchmark consists in analysing the system and the various categorisation dimensions to determine the benchmark category and measures of interest. The experimental dimensions (i.e., the operating environment, the workload, the faultload and the procedures and rules) are defined as a result of the definition of the benchmark category and features/measures of interest.

If, according to the various categorisation dimensions, comprehensive measures are of interest, a modelling step may be required in addition the experimentation. This is represented in Figure 4.1 that gives the various components and steps for system dependability benchmarking, at a very high level.

It is worth noting that, depending on the target system and on the measures of interest, the analysis step could be straightforward and may only consist in selecting the right benchmark category and measures as well as the various workload and faultload among the existing ones. However, for some target systems the analysis may be more elaborate. It may even consist in analysing deeply the system behaviour and even to prepare the modelling phase. In this latter case, the analysis may be time-consuming and may require a significant effort. Nevertheless, the results may be more complete than when only specific measures are considered. A trade-off may be made according to the objectives of the benchmarks, the effort required, and the results expected.

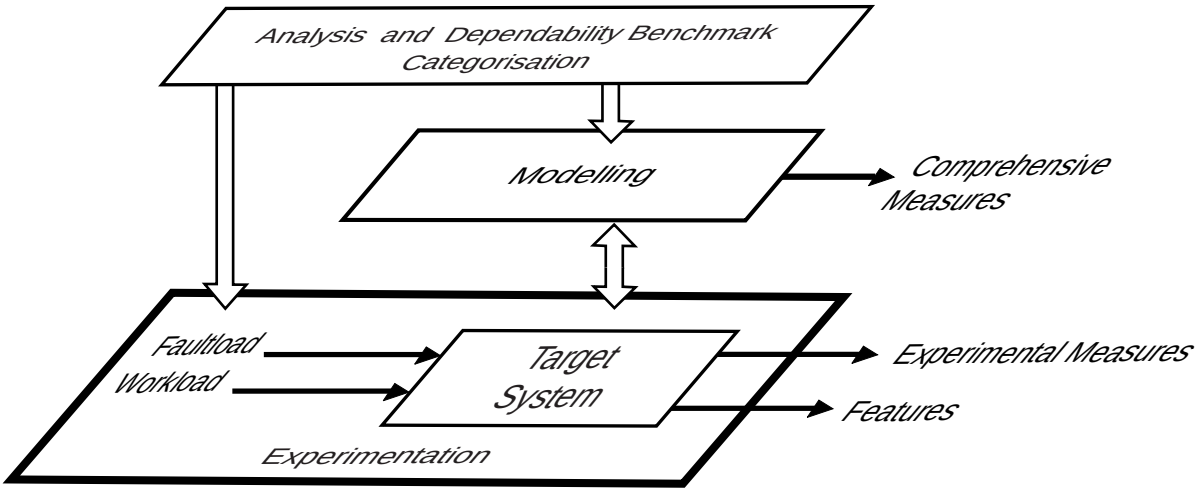

Figure 4.1: Dependability benchmarking overview

When the objectives of the benchmark are to identify the system capabilities and possible weaknesses, the final outputs may correspond to dependability features without necessitating any measure evaluation.

The link between measurement and modelling is discussed in the companion deliverable CF1, Section 2.4, related to the state of the art [CF1 2001]. In the current deliverable, we put emphasis on examples of scenarios for dependability benchmarking and we concentrate on the link between the inputs and outputs of modelling and experimentation in a benchmarking framework.

Whatever the complexity of the analysis step is, once the experimental dimensions and measures are identified, the experimentation can be performed according to the example of the scenario presented in Section 4.1.1. Sections 4.1.2 to 4.1.4 give three examples of benchmarking scenarios when modelling is involved, detailing the main links between modelling and experimental levels.

The first scenario illustrates the case where only system features and experimental measures are of interest, whereas the last two scenarios illustrate the case where comprehensive measures based on modelling are of interest as well.

4.1.1 Scenario 1: Dependability Benchmarking Based on Experimentation Only

The simplest scenario for benchmarking the dependability computer systems and components is actually an extension of the well-established performance benchmark setting. In this scenario, a dependability benchmark specifies a set of measures and all the experimental steps necessary to obtain these measures in accordance to the benchmark rules and conventions.

Comparing to the performance benchmark paradigm, two new important components are required (see also Figure 4.2).

- Features and Experimental measures – These are the outputs of the dependability benchmarking process. Features characterise qualitatively the system behaviour. The

experimental measures characterise the target system or component in an end-to-end way, taking into account the response of the target system during the experiment defined in the benchmark (e.g., availability or reliability measured during the experiment, performance measured in the presence of faults, detection coverage factor, etc.).

- Faultload – Consists of a set of faults and stressful conditions that are intended to emulate the real exceptional situations the system would experience in the field. The dependability benchmarks must specify the faultload (in principle, by describing the characteristics and statistical distribution of the different classes of faults) and the way the faults should be injected in the target system.

Figure 4.2: Simplest dependability benchmark scenario

Before discussing the technical problems raised by these new components, that constitute important research issues of WP2 and WP3, let us review the assumptions of the performance benchmark paradigm that could be shared with this dependability benchmarking scenario:

- There is a clear separation between the definition/adoption of benchmarks and the actual use of the benchmarks. Benchmarks are adopted by computer industry councils and represent industrial standards (or at least a general agreement among vendors or users). Once available, benchmarks can be used in two different ways:
  - As a standard – The benchmark results constitute standard results and, to attain this goal, the benchmark specification cannot be modified by the user (beyond the changes or adaptations stated in the benchmark documentation) and must be carefully respected. Results produced this way are useful for system/component characterisation as laid down by the standard (i.e., the benchmark) and for product comparison.
  - As a tool – The benchmark specification can be freely adapted to specific dependability evaluation needs. Results obtained this way do not represent standard benchmark results but can be used internally to improve system architecture, tune different configurations, or for training purposes.
- It follows a pure experimental approach and all the benchmark results are measured directly from the target system (the classical "what you get is what you measure").
- Measures tend to favour the "black-box" view of the target system/component. Typically, the target system/component is described in a functional way through the workload and few assumptions on the internal target architecture are stated in the benchmark specifications. This approach favours the end-to-end view of the target system/component and can be used even when the target documentation is not completely available. This is a particularly relevant advantage in the context of DBench, as COTS and COTS-based systems are the main targets for future dependability benchmarks.

- Benchmark representativeness is a crucial aspect, as benchmark results must characterise the addressed aspects of the target system (dependability, performance, both, other) in a realistic way. Concerning established performance benchmarks, the problem is reduced to the representativeness of the performance measures and the workload. Dependability benchmarking adds two new components to the problem, as it is necessary to define representative dependability measures and representative faultloads. These new problems constitute central research issues of DBench. Although the problem seems clearly more complex for dependability benchmarks than for performance benchmarks, the pragmatic approach used in the established performance benchmarks may also help us to find adequate solutions for dependability benchmarking representativeness. In short, the classical approach to assure representativeness in performance benchmarks consists of the following:

    - Performance benchmarks are targeted to specific application areas or specific computer components. This way the definition of representative performance measures is greatly simplified (because the representativeness domain is reduced).

    - Representative workloads are defined (agreed) either from well-established applications (i.e., real programs) or by defining a standard application as a set of tasks considered representative of the addressed application area or component.

    - Performance benchmark representativeness is a best effort rather than a "perfect solution". In practice, the main ingredients of the solution are previous experience of the field and agreement on what is considered representative for the computer industry.

In spite of the conceptual simplicity of this dependability benchmark scenario, many technical problems need in fact to be resolved. This challenge is being addressed in the DBench project through the tentative definition of benchmark prototypes (Chapter 5 presents some examples of tentative benchmarks that actually follow the dependability benchmarking scenario proposed in this section) and by research activities focused on specific problems. The following points summarise crucial research issues directly related to this dependability benchmarking scenario (and that will be addressed in WP2 and WP3):

- Definition of meaningful measures. Unclear aspects related to these measures are:

    - Representativeness of measures, in particular the experiment measures.

    - The way these measures can be obtained form actual target systems/components.

    - The possible impact of the measurements on the target system behaviour (intrusiveness).

    - The confidence and accuracy of the measurements.

- The adoption of the workloads of established performance benchmarks is the starting point for the definition of workloads. However, several problems still persist:

    - It is not clear whether the way the application spectrum has been partitioned by the performance benchmarks is adequate for dependability benchmarks or not.

  - Available workloads have to be expanded to take into account preventive and corrective maintenance actions that constitute routine operations for a given application area.
  - Definition of synthetic workloads for benchmarking specific computer components.

- The definition of real faultloads encompasses the following specific problems, which are currently being studied:
  - Definition and characterisation of the different classes of faults and stressful conditions that must be considered. Field data and previous experience are the main starting point sources.
  - Definition and analysis of the mapping between a set of erroneous behaviour and the typical set of fault classes.
  - Assignment of relative percentages to the different classes. Again, available field data and previous experience play a key role in the solution of this problem. The definition of faultloads that address just one class of faults seems to be another promising approach (e.g., hardware faults, operation faults, etc.), as the idea of benchmarking a system against a single class of faults seems useful.
  - Develop methods and techniques to emulate each class of fault.
  - Define the mechanisms and instrumentation required to introduce the faultload in the target system in a way that can be acceptable for a benchmark (i.e., cost-effective, easy to use, non-intrusive, etc.).

- Dependability benchmarks must meet some properties to be valid and useful. The complete set of benchmark properties is discussed in Section 4.2. However, concerning the dependability benchmarking scenario proposed in this section, one key property is reproducibility. In fact, this dependability benchmarking scenario is only possible if results can be repeated and reproduced by another party. Considering the statistical approach required for the faultload and the complexity of the system behaviour in the presence of faults, the fulfilment of the reproducibility property could be quite difficult.

4.1.2 Scenario 2: Benchmarking Dependability Using Modelling as a Support for Experimentation

This scenario differs from the previous one by the fact that the experimentation is guided, at least partially, by modelling (Figure 4.3). Actually, the constructed dependability model is processed and sensitivity analyses with respect to parameter values are made in order to identify the most significant parameters of the model (i.e., associated to the most salient features of the target system) that need to be evaluated accurately by experimentation. In this case, modelling helps in selecting the features and measures of interest to be evaluated experimentally as well as the right workload and faultload.

The outputs of scenario 2 are similar to those of scenario 1: features and experimental measures.

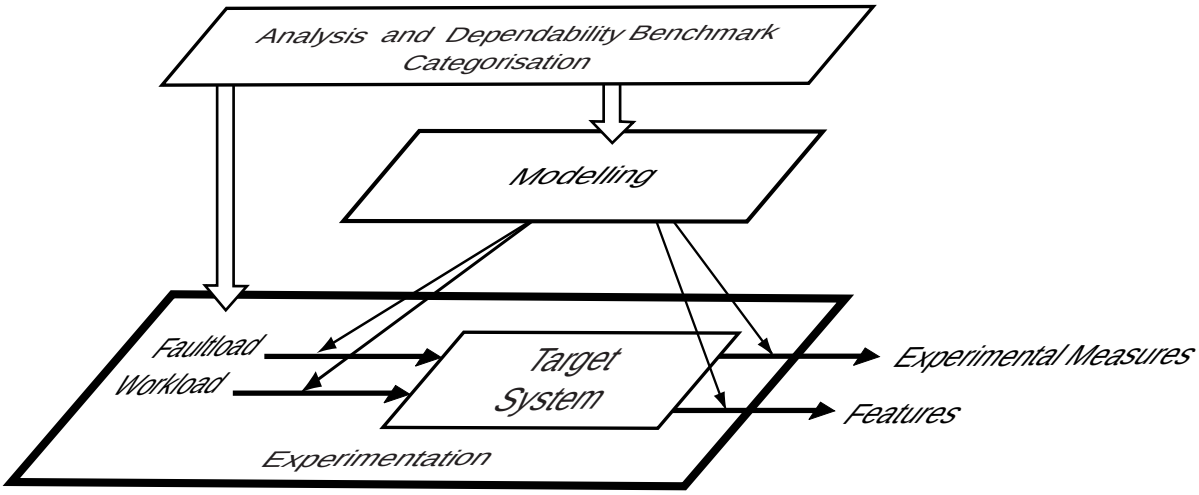

Figure 4.3: Modelling as a support for experimentation

4.1.3 Scenario 3: Benchmarking Comprehensive Dependability Measures Using Experimentation as a Support for Modelling

When comprehensive measures based on modelling are of interest, some specific dependability measures, used in the analytical model representing the system behaviour, may be provided by experimentation as shown in Figure 4.4. In this case, not only the selection of the features to be identified or the measures to be assessed experimentally are guided by modelling, but also the features identified during the experimentation may impact the semantic of the model. Therefore, the experimentations support model validation and refinement.

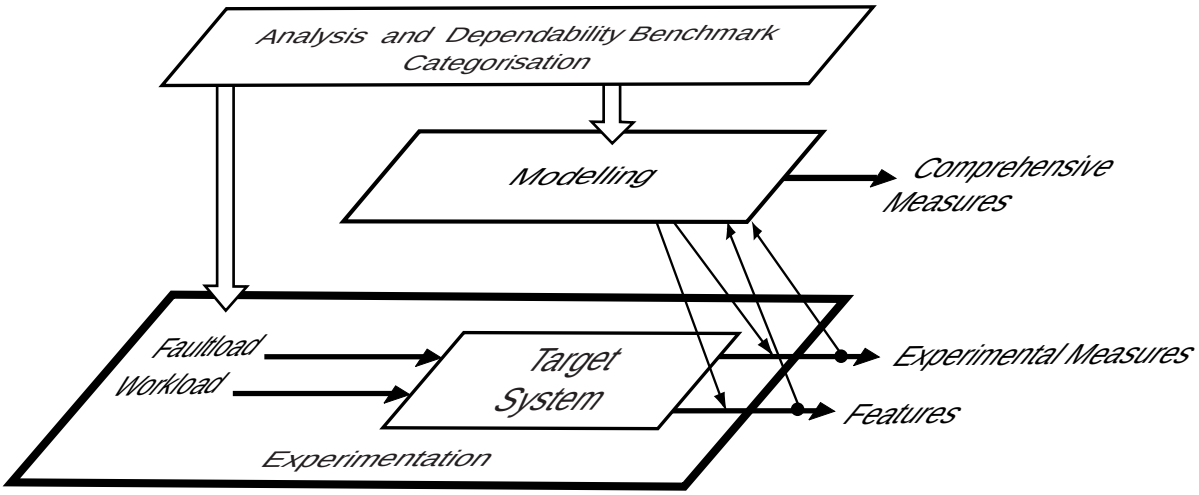

Figure 4.4: Experimentation as a support for modelling

Actually, model validation is a key step in dependability modelling and evaluation. In addition to experimental results, model validation is achieved by sensitivity analyses with respect to the assumptions made to construct the model and with respect to the parameter values. Such analyses increase our confidence in the model.

Usually, the analytical model requires additional parameter values that can be provided from field data or based on past experience related to similar systems.

The outputs of scenario 3 are comprehensive measures based on modelling (even though some features and specific experimental measures may be made available from the same experimental framework).

4.1.4 Scenario 4: General Dependability Benchmarking Framework Based on Modelling and Experimentation

In this scenario, the experimentation and the selection of features and measures of interest are guided, at least partially, by modelling, and modelling is supported by the experiment results. This scenario, given in Figure 4.5, is the combination of the two previous ones and it is the most complete one.

As for scenario 3, model validation is a key step in dependability modelling and evaluation.

The outputs of scenario 4 are: features, experimental measures, and comprehensive measures based on modelling.

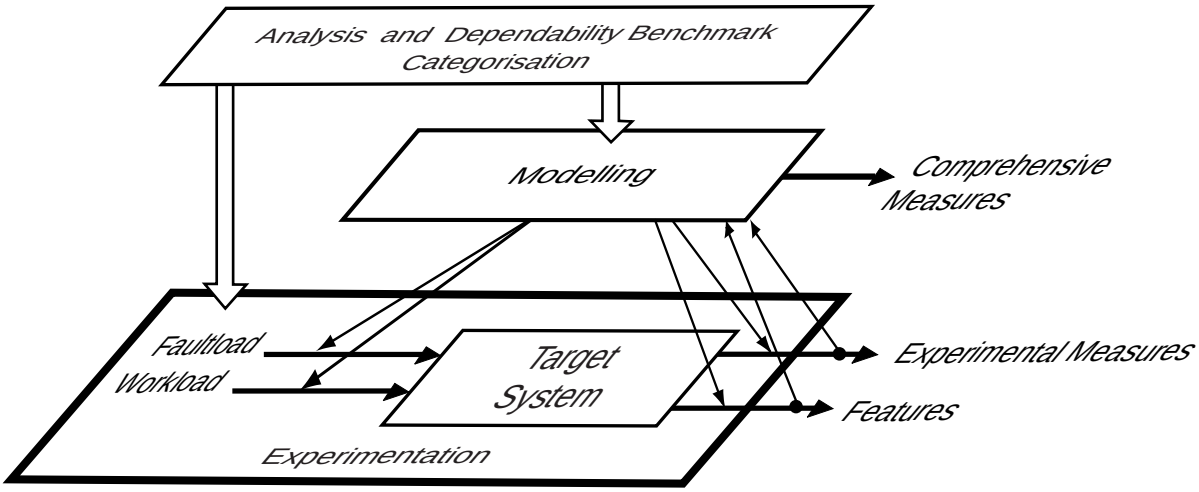

Figure 4.5: General dependability benchmark framework

4.1.5 General Discussion of the Different Scenarios

The described scenarios go from the simplest one (scenario 1) that can be seen as an adaptation of the traditional performance benchmarks to the more complete one in which modelling and experimentation are tightly coupled. Indeed scenario 1 is included in the three other scenarios. Additionally, all the problems and research issues presented and discussed for scenario 1 are also relevant for the other scenarios. The main difference between the four scenarios is related to the relationship between modelling and experimentation, and the evaluation of comprehensive measures in addition to measures obtained from experimentation.

The effort required and the benchmarking complexity increase from the simplest scenario (scenario 1) to the most complex one (scenario 4) while more results can be obtained from the most complex scenario.

The analysis step and the modelling step may be time consuming for complex systems requiring detailed modelling. However, for some systems, mainly COTS-based systems, for which the details about the system architecture may not be known, only high level, simple, and not detailed models can be constructed. In these cases, the resulting models may be not very complex and the modelling phase may require only a reduced effort leading to a trade-off between obtaining comprehensive measures characterising globally the system and accounting for the fault distribution of the faultload, with an acceptable effort (or cost).

The benchmark properties such as portability, cost, automation and execution time (that will be addressed in detail in Section 4.2), are easier to be satisfied for the less complex scenario than the more complex one.

4.2 Benchmark Properties

In addition to the dependability benchmark dimensions discussed in Chapter 3, that characterise the place of each specific benchmark in the multidimensional dependability benchmarking space, the benchmark properties represent another important set of dimensions that characterise the requirements (in the form of properties) that must be meet by each dependability benchmark to be considered valid and useful. For example, a benchmark must be repeatable (in statistical terms), must be representative, must not cause undesirable interferences with the target system, must be portable, cost effective, etc.

The relevance of the benchmark properties is quite clear, as they take into account of all the relevant problems that must be solved to define and validate actual dependability benchmarks. In this sense, the list of most relevant benchmark properties presented in this section summarises the main research goals of the DBench project, which constitute research topics planned for WP2 and WP3.

4.2.1 Reproducibility

For a benchmark to be credible it has to be reproducible. Reproducibility is the property, which guarantees that another party obtains statistically equivalent results when performing the same benchmarking activity. This property is central to benchmarking and is one of the foundations upon which the entire concept of benchmarking relies. Without reproducibility no one would be able to trust the results obtained from benchmarking activities. This would mandate that everybody used their own evaluation method, contradictory to the underlying philosophy and aim of benchmarking. All credible benchmarks must therefore be reproducible.

The problem of reproducibility has been addressed both in the areas of performance benchmarking as well as fault injection. The experiences from these areas will be used in the DBench project, but additional research is planned for the project to understand the conditions required to achieve reproducible dependability benchmarks.

4.2.2 Workload Representativeness

The workload representativeness reflects how well the workload of the benchmark corresponds to the actual workload of the real system. The workload representativeness must be considered in the context of the benchmark application area. Different areas have different workloads and may therefore need different benchmarks.

The experience from performance benchmarking will be quite helpful in achieving workload representativeness, and workloads available from performance benchmarks will be used as a starting point. Nevertheless, many issues still need to be researched concerning the definition of representative workloads for dependability benchmarks. Indeed, the erroneous behaviour of the system results from the combined effects of the workload and the faultload, and this interaction must be better understood. Other aspects related to workload representativeness will also be investigated in DBench, such as how to take into account preventive and corrective maintenance actions that constitute routine operations for a given application area and clearly affect the workload profile.

4.2.3 Faultload Representativeness

A faultload is composed of two parts, faults and stressful conditions. To achieve faultload representativeness both the set of faults injected and the stressful conditions must be representative.

Fault representativeness is a measure of how well injected faults correspond to real faults, i.e., faults affecting systems in real use. It is important to know how representative the faults are for the target system being benchmarked, so that it is possible to estimate how relevant the result is. This property must of course be considered in the context of the application area and the operating environment.

Stressful conditions are introduced via the workload and may cause errors. To increase faultload coverage it might be necessary to introduce stressful conditions that compromise the workload representativeness.

Faultload representativeness is one of the major concerns in DBench and an entire task (T22) of WP2 is devoted to the research of this problem.

4.2.4 Portability

Portability is how easily a benchmark can be extended to run on various systems within a particular application area. A portable benchmark can run on many systems within the application area. Portability enables the use of benchmarks for comparing computer systems or components.

Experiences from software development show that modularising software and using standardised high-level programming language, such as ANSI C increases portability. Avoiding the use of specialised hardware or software features make the benchmark more portable.

The achievement of the necessary portability for dependability benchmarks is complicated by the need of injecting faults and observing the faults effects in the target system at different levels. Research planned concerning enabling technologies (WP2), and particularly the extension of fault injection and instrumentation techniques, will address this specific problem.

4.2.5 Intrusiveness

If the benchmark requires modifications of the underlying system, it is intrusive. Intrusion is often costly and reduces portability and should therefore be avoided, especially physical intrusion.

Since the DBench project targets COTS and COTS-based systems, low intrusiveness approaches are required to inject faults and instrument the target systems. This is naturally another research topic in DBench concerning enabling technologies (WP2).

4.2.6 Interference

Interference is the undesired influence on the system caused by the benchmark. There are similarities between interference and intrusion, the difference being that interference is indirect while intrusion directly affects the underlying system. One example of interference is the possible effects upon scheduling introduced by real-time system benchmarks.

Per definition, fault injection interferes with the target system. Consequently, interference is unavoidable for dependability benchmarks making use of fault injection. A distinction has to be made between desired and undesired interference caused by the benchmark. Undesired interference can affect the result in unpredictable ways. Thus, it is vital that measures are taken to minimise the impact of undesired interference.

Dependability benchmarks require non-trivial instrumentation mechanisms to achieve good observability. These mechanisms might also introduce substantial interference. Finding suitable methods to tackle this problem is another major goal of the project.

4.2.7 Observability

When designing a benchmark, methods for probing the system have to be selected. Different probing methods have different advantages and disadvantages. Observability is the effectiveness of the instrumentation mechanism for probing the system state.

The choice of benchmark measures will have a direct impact on observability. Different measures require different probing methods. Intrusiveness, interference, execution time and the granularity of the results will be influenced by the choice of probing methods.

4.2.8 Scalability

To make a benchmark usable for systems of varying sizes, it has to be scalable. Scaling is done by increasing or decreasing the impact of one or more components of the benchmark. It is important that a benchmark remains representative after it has been scaled.

Usually, scaling is achieved by defining a set of scaling rules. Scaling will mostly affect the workload but other components such as faultload may also have to be scaled.

4.2.9 Automation

Benchmarking involves many steps. It is therefore important to achieve high level of automation in this process. Automation in this context is the ability to carry out the benchmark procedure with as little human intervention as possible.

The automation property affects the benchmarking procedure mainly through its influence on cost, execution time and ease-of-use. To have a chance of becoming widely accepted, future dependability benchmarks have to offer a high level of automation. Experience from implementing automated fault injection tools will of course be valuable when constructing the benchmark toolset.

4.2.10 Execution Time and Cost

The execution time of a benchmark is the time required to obtain the result from the benchmark. The execution time consists of three parts: i) setup and preparations, ii) running the actual benchmark program and iii) data analysis. Of course it is desirable to have as short an execution time as possible.

Ideally one would construct benchmarks that both evaluate the system thoroughly and accurately, while having short execution times. Normally, these properties are contradictory which calls for a trade off. It might be possible to shorten the execution time by increasing the level of automation.

A key goal of dependability benchmarking is to provide an efficient and cost effective approach to characterise dependability of computer systems and components. Of course, dependability benchmarking is only attractive as long as its perceived value is higher than the associated costs

5. Some Examples of Preliminary Dependability Benchmarks

DBench addresses three categories of systems: transactional systems, embedded system and operating systems. In this chpater, we present our very first thoughts and plans for benchmarking transactional systems and operating systems. Of course, we can only give a high level of the work that will be done mainly in WP3 according to the results obtained in WP2. However the selected examples will give concrete ideas about our approaches to benchmarking these categories of target systems. This current view may evolve and will be refined along the project.

5.1 Dependability Benchmark for Transactional Applications

The goal of this section is to propose preliminary dependability benchmark for transactional applications. This dependability benchmark is particularly targeted to OLTP (On-Line Transaction Processing) systems, which constitute the kernel of the information systems used today to support the daily operations of most of the business. Some examples include banking,

insurance companies, all sort of travelling businesses, telecommunications, wholesale retail, complex manufacturing processes, patient registration and management in large hospitals, libraries, etc. These application areas comprise the traditional examples of business-critical applications, which clearly justify the interest of choosing them as target for the first dependability benchmark for transactional applications.

This preliminary dependability benchmark follows the benchmarking scenario presented in Section 4.1.1. The experiment measures characterise both system dependability and performance from the end-user point of view. Some of these measures can be used later on in more complex dependability benchmarking scenarios, such as those described in Section 4.1.2, Section 4.1.3, and Section 4.1.4.

It is worth noting that the measures proposed for this preliminary dependability benchmark follow the well-established measuring philosophy used in the performance benchmark world. In fact, the measures provided by existing performance benchmarks give relative measures of performance (i.e., measures related to the conditions disclosed in the benchmark report) that can be used for system/component improvement and tuning and for system comparison. It is well known that performance benchmark results do not represent an absolute measure of performance and cannot be used for planning capacity or to predict the actual performance of the system in field. In a similar way, the measures proposed for this first dependability benchmark must be understood as benchmark results that can be useful to characterise system dependability in a relative fashion or to improve/tune the system dependability.

The proposed set of measures has the following characteristics/goals:

- Focus on the end-user point of view (real end-user and database administrators).
- Focus on measures that can be derived directly from experimentation (i.e., following the general tradition of the performance benchmark world that relies on direct measures).
- Include both dependability and performance measures.
- Measures must be easily understandable (in both dependability and performance aspects) by database users and database administrators.

5.1.1 Dependability Benchmark Environment

The key elements of the experimental environment required to collect the measures are represented in Figure 4.6.

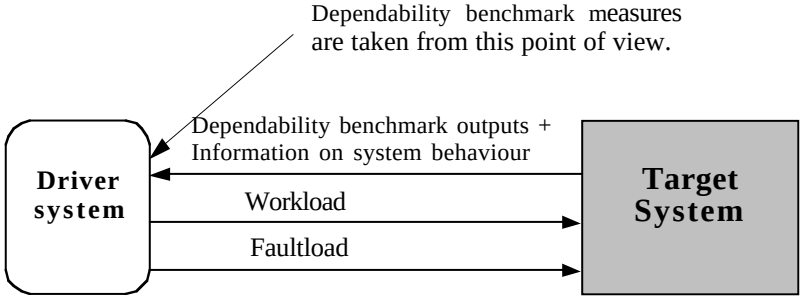

Figure 4.6: Key elements of the experimental environment.

The measures will be obtained through a set of experiments. The target system will run a workload and the experiments include two main phases. In a first set of experiments the workload is run without (artificial) faults and in a second set of experiments the same workload is run with a faultload. Finally, a set of consistency tests (in addition to the normal consistency texts included in the workload) can be performed on the data processed by the transactions to detect possible integrity violations in the final data resulting from the transactions.

The target system represents a client-server system (either a traditional client-server or a three tier system) fully configured to run the workload. A large variety of architectures can be used ranging from centralised systems to different configuration of parallel and distributed systems. From the benchmark point of view, the target system is the set of processing units used to run the transactional workload and to store all the data processed by the transactions.

Considering the application area addressed, our first choice is to adopt the workload of the well-established TPC Benchmark™ C (TPC-C). This workload (see also CF1, Chapter 5 [CF1 2001]) represents a mixture of read-only and update intensive transactions that simulates the activities of most OLTP application environments, including transactions resulting from human operators working on interactive sessions.

The TPC-C workload also includes an external driver system, which emulates all the terminals and their emulated users during the benchmark run. All the performance and dependability measures are collected from the point of view of the emulated users. In other words, the measures correspond to an end-to-end characterisation of performance and dependability from the end-user point of view.

The faultload represents a set of faults and stressful conditions that simulate real faults experienced by OLTP systems in the field. Although it will be desirable to emulate a comprehensive range of classes of faults, it seems useful and interesting to concentrate first on specific classes of faults. In this context the most interesting faults for OLTP system are the operational faults. These faults include human (administrator and operator) errors and environment faults such as power failures, disconnection of cables, etc. Several reasons account for the decision of exploring first the operational faults:

- Data from field, especially from the maintenance service of major database vendors, rate administrator errors as the first cause of OLTP systems outages.
- A representative set of these faults can be specified from field data and from interviews of experienced database administrators.
- The emulation of these faults can be carried out by specific (and small) programs running on the target system, emulating common administrator mistakes. In this way, it is not required to use fault injection tools that may complicate the implementation of the benchmark. Most of the environment faults can also be easily introduced in the target system. The activation of the operational faults can be controlled by the same driver system used to emulate the interactive user.

Other faultloads including the emulation of hardware and software faults will be considered after getting some feedback from using a faultload based only on operational faults. At the same time, results from current research on fault representativeness [Durães *et al.* 2001,

Jarboui 2001]), that is being done in the context of WP2, will also be used to define more comprehensive faultloads for this dependability benchmark.

5.1.2 Dependability Benchmark Measures

The measures must be taken in the different experiment phases shown in Figure 4.7.

- Phase 1 - First run or set of runs of the workload with no faultload. This run will be used to collect the baseline performance measures. These performance measures represent the performance of the system with normal optimisation settings. The idea is to use them in conjunction to the other measures (see below) to characterise system dependability (and not as a measure of system performance).

- Phase 2 – In this phase the workload is run in the presence of faults with the goal of measuring the impact of faults on the performance. The idea is to run the workload during an extended period of time (e.g., 8 hours or more) and inject the faults specified in the faultload. After injecting one fault the target system may continue executing transactions normally if nothing wrong is detected. Otherwise, the target system may start a recovery procedure, depending on the fault effects and the detection and diagnosis mechanisms in the system, or may need to be restarted completely if recovery is not enough. The driver system will restart the database system after a fault whenever the establishment of client sessions with the target database fail. The complete restart of the target systems (i.e., operating system + database) will be performed whenever the driver system detects a target system crash. An important goal is to assure that phase 2 is completely automatic and can be performed without user intervention.

- Phase 3 – This last phase consists of running a set of application consistency tests (specified by business rules in the TPC-C specification) to check possible data integrity violations. These integrity checks will be performed on the application data (i.e., the data in the database tables after running the workload) and will use both business rules (defined in the TPC-C specification) and the database metadata to assure a comprehensive test. Possible integrity violations detected in this phase represent application data errors from the end-user point of view. It is worth noting that these tests are performed in addition to the normal integrity test included in the workload and in the database engine that may also detect integrity errors during phase 2.

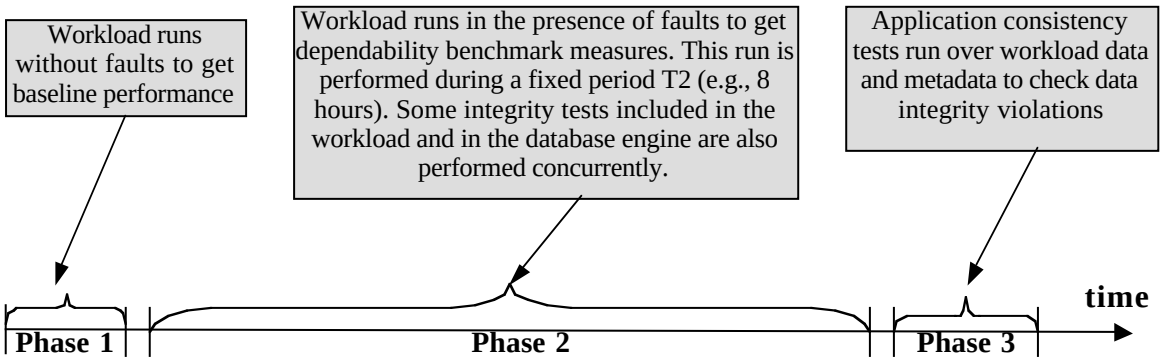

Figure 4.7: Dependability benchmark for transactional applications

Some examples of dependability benchmark measures that can be collected during the three phases are the following:

Tb Baseline transactions per minute. This measure corresponds to the classical TPC-C performance measure known as tpmC. However, Tb represents a baseline performance instead of an optimised performance measure (as is the case of tpmC) and should consider a good compromise between performance and recoverability.[4]

$/Tb Price per transaction. The price is established using a set of pricing rules specified in the TPC-C specification.

Tf Number of transactions executed in the presence of the faults specified in the faultload during the time T2 (see Figure 4.7). It measures the impact of faults in the performance and favours systems with higher capability of tolerating faults, fast recovery time, etc.

$/Tf Price per transaction in the presence of faults of the faults specified in the faultload during the time T2. It measures the (relative) benefit of including fault handling mechanisms in the target systems.

Tf/Tb System performance decreasing ratio due to the faultload.

Ta Number of transactions aborted because of the faults. It measures the impact of faults in the form of lost transactions or transactions that must be resubmitted. It favours systems with efficient recovery mechanisms.

Ne Number of errors detected by the workload consistency tests and metadata tests. It measures the impact of faults on the data integrity.

Avt Availability during the test period T2. It measures of system availability during the test period T2 from the end-user point of view. This measure corresponds to Avt = Tav/T2, where Tav is the total time the system is available from the point of view of the end-user and T2 is the testing period of phase 2. Several classes of availability can be considered as a specialisation of this measure. For example Avt can represent the case when all terminals are unavailable; Avt50 the availability when less than 50% of the terminals are unavailable; Avt10 the availability when less than 10% of the terminals are unavailable, etc.

In addition to the research work that is being conducted in the framework of WP2, that addresses fundamental issues such as faultload and workload representativeness, several specific experiments are already being carried out to evaluate the feasibility of this dependability benchmark proposal [Vieira *et al.* 2001].

---

[4] TPC-C performance measure (tpmC) is obtained by configuring all the database parameters to get the maximum performance, which normally deteriorates the recovery features of the database.

5.2 Dependability Benchmark for Operating Systems

As it orchestrates the main interactions between the hardware platform and the application software layers, as well as the main scheduling of the application tasks, the Operating System (OS) is the core component for any computer system, being it intended for classical transaction-based applications or embedded into specific applications areas, including those with high dependability requirements. Accordingly, much attention must be paid to the characterisation of such a component whose malfunctions and failure modes have a strong impact on the dependability of the global system.

In addition, so as to cut with development costs and still to master the inherent complexity of the services to be provided by an OS, the tendency is to use more and more off-the-shelf components (either commercial or open source). It follows that the system integrator is faced with the following two major sets of questions:

1) Selection: What are the failure modes of the available operating systems? Which OS(s), among the wide variety of the offer of commercial or open source OSs, would be suitable with the dependability requirements imposed by the application?

2) Evaluation-based design: How to progressively assess the dependability of the system in which the candidate OS is being integrated?

On the other side of the spectrum, the OS provider is of course attentive to monitor the dependability characteristics of the product he is being developing and commercialising.

Accordingly, specific benchmarks need to be developed for assessing the dependability characteristics of OS components.

In the DBench project, we will mostly consider the **integrator perspective** for what concerns benchmarking OSs. This means that it will be assumed that the information and documentation available on the target OSs is either absent, or least sparse or incomplete. Accordingly, emphasis will be put on the "black-box" view that is attached to COTS components. This also means that the benchmarks being developed should not rely on detailed information on the architecture and implementation of the target systems.

The main goal pursued by the work concerning the benchmarking of OSs will be devoted to significantly advance the state-of-the-art with respect to the previous related research already carried out. One major improvement that is anticipated will concern the reaching of a more standard form of benchmarking, with respect to both the definition of the benchmark components and the conducting of the related experiments and measurements.

Building upon the proposed benchmarking framework, as well as upon the generic model described earlier in this chapter, two main facets of a benchmark will be considered successively in the sequel: i) the input domain (the faultload and workload components) and ii) the output domain (the different sets of measurements and measures). Accordingly, what follows can be seen as an instantiation of the proposed benchmarking framework and model to the OS area.

5.2.1 On the Input Domain

Towards this ends, the robustness testing techniques where the target OS is subjected to erroneous and/or stressful service/kernel calls, (see CF1, Chapter 3 [CF1 2001]) offers a promising approach to start with. In order to better identify the **faultload** component of the benchmark to be considered, we will build up on the results of the extensive set of experiments concerning fault representativeness, currently being conducted on off-the-shelf components in the framework of WP2 (e.g., see [Durães *et al.* 2001, Jarboui 2001]), and especially those focusing on Linux and Windows 2000 OSs. Both deterministic (application of specific fault/error patterns on specific kernel calls) and statistical (randomly selected corruptions on randomly selected calls and associated parameters) strategies will be considered for the application of the faultload application. In order to facilitate the acceptance of the benchmark prototypes that will be investigated, we will try to rely as much as possible on well identified Application Programming Interfaces (APIs) and whenever possible official or *de facto* standard APIs (e.g., POSIX) to precisely specify the faultload. Indeed, this is a prerequisite to facilitate the portability of a benchmark to assess and compare several OS candidates on a fair basis.

Concerning the **workload** component of the benchmark, we will first consider synthetic workloads. Moreover, the workloads will be tailored to focus the main functional components that support the central services offered by an OS. In particular, the following functional components will be distinguished:

- Synchronisation: semaphore management for process synchronisation.
- Scheduling: creation, scheduling policies and termination of processes; interrupt and time management.
- Memory management: management of addressing spaces, execution modes, mapping.
- Communication: inter-process communications and remote process calls.
- File system: management of peripherals and file formats.

5.2.2 On the Output Domain

An important dimension of a benchmark output concerns the types of **measurements** that are carried out and the **measures** that are provided to the user in order to characterise the faulty behaviour of an OS. Three major forms of reactions of the target OS need to be monitored:

- Value Failures: these encompass erroneous data observed at the levels of the API (e.g., inconsistent call return) or of the workload (e.g., erroneous behaviour or wrong output result).
- Timing Failures: these encompasses either early or late timing failures (e.g., kernel hang, workload hang or deadline miss, etc.)
- Error Detection/Signalling: these correspond to the error checks that are provided by an OS (e.g., exceptions, error status signals, etc.).

Whenever possible, both occurrences and related timing measurements will be recorded for the above outcomes.

The modular decomposition of the workload and of the set of experiments allow for a more precise information on the functional components. In particular, it allows for the error propagation channels to be identified and evaluated. This is actually very much important from a design-aid viewpoint not only for the integrator, but also for the OS provider as well. Indeed, such analyses are very much useful to devise suitable architectural solutions (including wrappers) to improve the failure modes and robustness of the candidate OS to be integrated into a dependable computer system.

Of course, such refined measures can also be used to derive a more comprehensive assessment of the faulty behaviour of the target OS. Indeed, an overall system-level picture can be conveyed by combining the results obtained from the various workload processes applied.

6 Conclusion

The goal of dependability benchmarking is to provide generic ways of characterizing the behaviour of components and computer systems in the presence of faults, allowing for the quantification of dependability measures. Beyond existing evaluation techniques, such as analytical techniques and experimental approaches based on fault injection and robustness testing, dependability benchmarking must provide a uniform, repeatable and cost-effective way of performing this evaluation either as stand alone assessment or more often for comparative evaluation across systems.

This deliverable presents the preliminary framework proposed by the DBench project to investigate, define, and validate dependability benchmarks for computer systems, with particular emphasis on COTS and COTS-based systems.

Given the relevance of dependability measures (the actual expected result of dependability benchmarks) we first presented a framework for the definition of dependability benchmark measures. Two main groups of dependability measures have been identified: comprehensive measures (e.g., reliability, availability, maintainability, and safety) and specific dependability measures (e.g., error detection efficiency, error recovery efficiency and error containment efficiency). Comprehensive measures characterise the system globally at the service delivery level (but note that a system could be a component of a larger system), taking into account all events impacting its behaviour and their consequences. Specific measures characterise particular aspects of a system or a component: behaviour in presence of fault, maintenance, and system evolution/upgrade. Comprehensive measures are usually evaluated based on analytical modelling and some of the specific measures may appear as parameters in the analytical model. It is worth noting that the comprehensive dependability measures, when evaluated based on modelling, represent something new in the context of benchmarking, as existing performance benchmarks only rely on direct measures taken at the time of a specific benchmarking event.

Great care has been put on the identification of the benchmark measures that can be obtained experimentally. In addition to the specific dependability measures, which can be evaluated through an experimental benchmarking process, we identified a new type of comprehensive measures that can be obtained directly from benchmark experiments. Examples of these

measures are system availability, reliability, or number of integrity errors observed during the benchmark running, while the system is working under a standard experimental environment (i.e., standard workload, standard faultload, etc). Although these comprehensive measures are intimately related to the standard experimental environment used in the benchmarking process (i.e., cannot be confused with unconditional system availability or reliability, for example), they can be quite useful to help to improve systems/components or to compare different systems/components. Additionally, as performance is intimately related to dependability, two kinds of performance measures are also relevant to dependability benchmarking: the baseline performance (i.e., performance in absence of faults) and the performance in presence of faults, to characterise the performance degradation due to fault activation when compared to the baseline performance.

Due to the multi-dimensional nature of the problem, the definition of a framework for dependability benchmarking requires first the identification and the clear understanding of all the dimensions of the problem. In addition to measures, which represent a key dimension, three additional groups of dimensions have been identified and discussed. Categorisation dimensions (including the benchmark usage, the life cycle phase, the application area, and the target system nature) define in practice the general bounds of the problem and allow us to organize the dependability benchmark space into benchmark categories. A second group of dimensions is the experimental dimensions, which include all the aspects related to the experimental steps of benchmarking (i.e., the operating environment, the workload, the faultload, and the procedures and rules required to run a benchmark experiment). Finally, the dependability benchmark properties represent a set of dimensions stating the key properties that dependability benchmarks must meet to be valid and useful. For example, a benchmark must be repeatable (in statistical terms), must not cause undesirable interferences with the target system, must be portable, cost effective, etc. These benchmark properties are in fact very relevant, as they represent the main problems that must be solved to define and validate actual dependability benchmarks. In this sense, the long list of benchmark properties presented and discussed also summarizes the main research goals of the DBench project, which constitute current research topics in the project.

From the multi-dimensional analysis of the problem, the framework proposed for conducting dependability benchmarks encompasses several steps and possibilities. The first step consists in analysing the system and the various categorisation dimensions to determine the benchmark category and measures of interest. The experimental dimensions are defined as a result of the definition of the benchmark category and features/measures of interest. If, according to the various categorisation dimensions, comprehensive measures are of interest, a modelling step may be required in addition the experimentation.

To better understand the multiple possibilities of the proposed dependability benchmarking framework, we presented four different benchmarking scenarios, going from the simplest one (that can be seen as an adaptation of the traditional performance benchmarks) to the more complete one, in which modelling and experimentation are tightly coupled. The main difference between the four scenarios is related to the relationship between modelling and experimentation, and the evaluation of comprehensive measures in addition to measures obtained from experimentation.

The first scenario consists of dependability benchmarking based on experimentation only and illustrates the case where just system features and/or specific and special experimental measures are of interest. The second scenario is focused on benchmarking dependability using modelling as a support for experimentation and the idea is to guide the experimentation (at least partially) by modelling. The last two scenarios illustrate the cases where comprehensive measures are of interest as well: the first one consists of benchmarking comprehensive dependability measures using experimentation as a support for modelling, while the second represents the general dependability benchmarking framework based on modelling and experimentation. This scenario is the most complete one and actually includes all the possibilities addressed in the other scenarios.

Finally, two examples of real dependability benchmarks already under research in the project were presented. These examples follow the dependability benchmarking based on experimentation only and consist of a system wide dependability benchmark for transactional applications, which is one of the application areas of the project, and a dependability benchmark for operating systems, which are one of the most relevant components in a computer system. Although these examples represent just preliminary instantiations of the proposed framework, the evolution and validation of these first examples of dependability benchmarks will help us to focus the research that is being conducted in the framework of WP2 and WP3. In fact, the validation of these first examples as preliminary inputs for WP3 (in which the planned benchmark prototypes will be defined) is very dependent on the results that will be obtained in WP2, especially in what concerns the measurements to be performed on the target system, the representativeness of measures, workload, and faultload.

References

[Avizienis *et al.* 2001] A. Avizienis, J.-C. Laprie and B. Randell, *Fundamental Concepts of Dependability*, LAAS Research Report, N°1145, April 2001.

[CF1 2001] J. Arlat, K. Kanoun, H. Madeira, J. V. Busquets, T. Jarboui, A. Johansson, R. Lindström S. Blanc, Y. Crouzet, J. Durães, J.-C. Fabre, P. Gil, M. Kaâniche, J. J. Serrano, J. G. Silva, N. Suri and M. Vieira, "*Conceptual Framework, Deliverable CF1, State of the Art*", DBench Project, IST 2000-25425, August 2001.

[Costa *et al.* 2000] D. Costa, T. Rilho, and H. Madeira, "Joint Evaluation of Performance and Robustness of a COTS DBMS through Fault-Injection", *Proc. of Int. Conference on Dependable Systems and Networks (DSN-2001)*, (New York, USA), IEEE Computer Society Press, 2000.

[Durães *et al.* 2001] 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), pp. B.92-B.93, Chalmers University of Technology, Göteborg, Sweden, 2001.

[Folkesson & Karlsson 1998] P. Folkesson, S. Svensson and J. Karlsson, "A Comparison of Simulation Based and Scan Chain Implemented Fault Injection", in *Proc. 28th Int. Symp. on Fault-Tolerant Computing (FTCS-28)*, (Munich, Germany), pp.284-293, IEEE Computer Society Press, June 1998.

[Jain 1991] R. Jain, *The Art of Computer Systems performance Analysis — Techniques for Experimental Design, Measuremnt, Simulation, and Modeling*, 685p., John Wiley & Sons, Inc., 1991.

[Jarboui 2001] T. Jarboui, "Fault Models Consolidation: Linux as a Case Study", in *Supplement of the Int. Conference on Dependable Systems and Networks (DSN-2001)*, (Göteborg, Sweden), pp. A.13-A.15, Chalmers University of Technology, Göteborg, Sweden, 2001.

[Kaâniche *et al.* 2001] M. Kaâniche, K. Kanoun and M. Rabah, *A Preliminary Framework for SoS Dependability Modelling and Evaluation*, Deliverable DSoS Project, IST-1999-11585, LAAS Report, N°1157, April 2001.

[Kanoun *et al*. 1997] K. Kanoun, M. Kaâniche and J.-C. Laprie, "Qualitative and Quantitative Reliability Assessment", *IEEE Software*, 14 (2), pp.77-86, mars 1997.

[Koopman & DeVale 1999] 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), pp.30-7, IEEE CS Press, 1999.

[Labovitz *et al.* 1999] C. Labovitz, A. Ahuja and F. Jahanian, "Experimental Study of Internet Stability and Backbone Failures", in *Proc. 29th IEEE Int. Symposium on Fault-Tolerant Computing (FTCS-29)*, (Madison, WI, USA), pp.278-85, IEEE Computer Society Press, 1999.

[Laprie 1995] J.-C. Laprie, "Dependable Computing: Concepts, Limits, Challenges", in *Proc. 25th Int. Symp. on Fault-Tolerant Computing (FTCS-25). Special Issue*, (Pasadena, CA, USA), pp.42-54, IEEE Computer Society Press, 1995.

[Sinha & Suri 1999] P. Sinha and N. Suri, "Identification of Test Cases Using a Formal Approach", in *Proc. 29th Int. Symp. on Fault-Tolerant Computing (FTCS-29)*, (Madison, WI, USA), pp.314-321, IEEE CS Press, 1999.

[Vieira *et al.* 2001] M. Vieira, D. Costa, and H. Madeira, "Optimization of Performance and Recovery of Database Systems in The Presence of Operation Faults", in *Supplement of the Int. Conference on Dependable Systems and Networks (DSN-2001)*, (Göteborg, Sweden), pp. B102-B103, Chalmers University of Technology, Göteborg, Sweden, 2001.