DBench

Dependability Benchmarking

IST-2000-25425

Measurements

Report Version: Deliverable ETIE1

Report Preparation Date: June 2002

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 Technologies" Programme (1998-2002)

Contents

Measurements

Authored by: Karama Kanoun*, Yves Crouzet*, Jean Arlat* and Henrique Madeira**

With the contribution of all other project members

* LAAS ** FCTUC

21 June 2002

Abstract

This deliverable is devoted to the definition of measurements to be performed on a system under benchmark in order to characterise/benchmark the dependability of this system, i.e., assess particular dependability benchmark measures. We first briefly present an update on our current vision of the benchmarking dimensions. Then describe succinctly the classes of systems to be studied in DBench before focusing on examples of measures of interest to be assessed for each class of systems. Finally we address fault tolerance specific measures, putting emphasis on the kind of information to be collected and on how to present results to make them exploitable.

1 Introduction

This deliverable is devoted to the definition of measurements to be performed on a system under benchmark in order to characterise/benchmark the dependability of this system, i.e., assess particular dependability benchmark measures.

The dependability benchmarking framework, initially defined in Deliverable CF2 [Madeira *et al.* 2001], has identified the various dimensions influencing the definition and results of a dependability benchmark. These dimensions have been grouped into three groups namely: categorisation, measure and experimentation dimensions, in addition to benchmark properties that should be satisfied such as portability, usefulness and cost.

The *categorisation* dimensions allow us to organize the dependability benchmark space into well-identified categories by allocating a particular "value" to each dimension. The set of categorisation dimensions describe the system under benchmark and the benchmarking context. The *measure* dimensions specify the dependability benchmarking measure(s) to be assessed depending on the categorization dimensions. The *experimentation* dimensions include all aspects related to experimentation on the system under benchmark to get the base data needed to obtain the selected measure(s).

The combination of the various dimensions leads to a large variety of benchmarks. It is not possible to cover all of them in DBench. As a consequence, we will focus our analyses on a reduced set of benchmark classes in order to be able to reach pragmatic and useful results.

Even though this deliverable is specifically dedicated to measurements achieved on the system to assess the benchmark measures of interest, we need to specify, at least at a high level, the categorisation dimensions to define the measures of interest according to the benchmark category.

It is worth noting that, for sake of simplicity, the term "measure" is used in a broad sense to designate a quantitative measure or a qualitative feature of the system. For example, the failure modes are refereed to as measures of dependability that can be obtained from experimentation and measurements on the system.

With respect to the previous deliverables, we have refined the dimensions included in the three groups to better reflect real situations and have separated the descriptive concerns from experimental aspects. We will thus briefly present our current vision of these groups of dimensions (without redefining all concepts) before presenting the systems that will be studied in DBench.

The rest of this deliverable is organised as follows. Section 2 briefly presents an update on our current vision of the benchmarking dimensions. Section 3 describes succinctly the classes of systems to be studied in DBench. Section 4 presents the benchmark measure dimensions. Section 5 focuses on examples of measures of interest to be assessed for each class of systems, while Section 6 focuses on fault tolerance specific measures. Section 7 summarises the deliverable.

2 Benchmarking Dimensions

Figure 1 presents the three groups of dimensions.

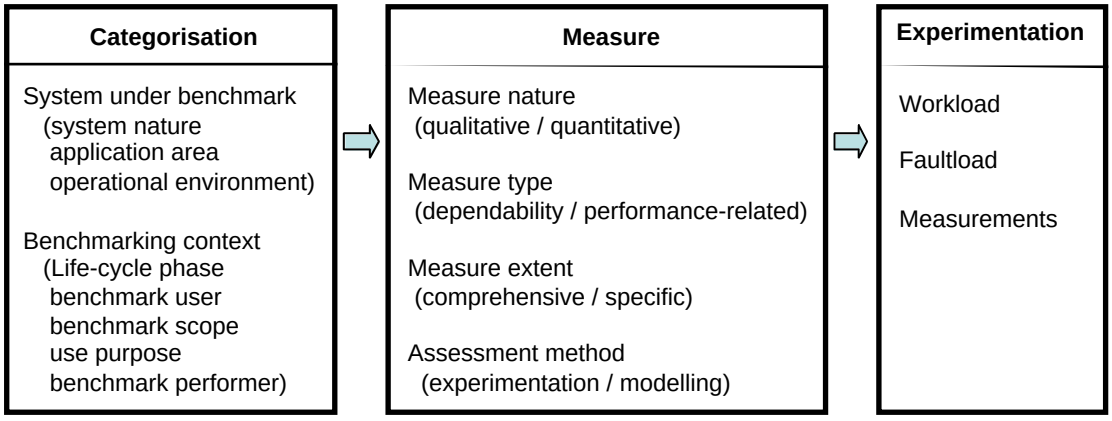


Figure 1: Summary of benchmark dimensions

The main changes for the categorisation group concern the fact that we separate the definition of the system under benchmark from the definition of the benchmark context. This separation of concern will allow us for example to benchmark the same system for various contexts or to compare several systems for the same context.

The system under benchmark is defined with respect to its nature, the application area, as well as its operating environment.

System nature - The system ranges from discrete hardware and software components to complete systems. The system can be presented according to two complementary views:

- An abstraction layer view, as a set of abstraction layers (or components) performing specific functions, including key components for dependability.
- A functional view, in terms of functionalities and features, including dependability features.

Application area - The application area is a key dimension for the definition of dependability benchmarks, as it impacts the system activation profile (i.e., the workload), the operational environment, the selection of benchmark measures, as well as the set of possible faults that could affect the system (and hence the faultload to be applied for benchmarking).

Operational environment - The operational environment characterizes the environment in which the system is used in operation. It gathers aspects such as maintenance actions and external resources required for system operation, including possible interactions with human operators and end-users or other systems.

The benchmarking context is defined clearly through the definition of the life cycle phase in which the benchmark is achieved, the benchmark user, the benchmark scope, the use purpose and the benchmark performer.

Life Cycle Phase - A dependability benchmark could be performed in any phase of the system life cycle from early development phases (i.e., design and specifications) to operation. The results could be used for the current or subsequent phases.

It is worth mentioning that components, particularly off-the-shelf components (OS or Data Base Management Systems), are benchmarked in their final product state, when they are available on the marked. The end-users can thus use benchmark results in their purchase decisions. Also, COTS components can be benchmarked during the integration phases of a system in which they are being integrated, either for internal purpose use or in a standard way, to be made available for external purpose use.

When developing complex systems, it is common that different entities collaborate to produce the final product, which is then delivered to the end-user. Complex systems are normally built by dividing the system into smaller parts, which can be designed and constructed separately. These components are then put together to form the complete system. This division of systems consisting of components can be applied through all levels of a design. The relationship between different entities will thus form a hierarchy in which higher-level entities integrate the work of lower level entities to form a system. This system may then be integrated as a component into an even more complex system. We call the lowest level in the hierarchy component developers, and the user of the final product the end-user. All levels in between are considered to be system integrators. It is worth noting that a system integrator at an intermediate level may interact directly with component developers and system integrators of lower levels. Also, in this hierarchical presentation, a system integrator a given level can be considered as a system provider for the higher levels.

Benchmark user - Person or entity actually using the benchmark results. The primary users are the *integrator of the system* or end-user. However, the benchmark results are to be communicated to the successive integration levels up to the *end-users* of the integrated system. Nevertheless, some results may be of interest to the designer(s) of the benchmarked COTS component, for improving the concerned component, in case the benchmark reveals some deficiencies.

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

Use purpose - Benchmark results are intended for or external or internal use purpose. External use purpose involves standard results that fully comply with the benchmark specification, for public distribution, while internal use purpose is mainly for system validation and tuning. Benchmarks for internal use purpose can be based on proprietary rules that do not need to be standardized. However, this does not mean that internal use purpose measures should not be made available for the business community. This only means that they cannot all be specified in a standard way due to the various particularities of systems within the class of systems

addressed. Nevertheless, it is recommended to communicate the results together with their precise definition and conditions under which they have been obtained, in order to be useful.

The benchmarks considered in DBench are primarily meant for external use purpose. However, we will also define measures that are meant for internal use purpose and give recommendations on how to make them exploitable by other entities.

Benchmark performer - Person or entity performing the benchmark (e.g., manufacturer, integrator, third party, end-user). These entities have i) different visions of the system under benchmark, ii) distinct accessibility and observability levels for experimentation and iii) different expectations from the measures. The benchmark performer and the benchmark user could be the same entity, but could also be different. We consider that the benchmark performer is someone (or an entity) who has no in depth knowledge about the system under benchmar and who is aiming at i) improving significantly its knowledge about its dependability features and ii) publicizing information on the dependability of the system under benchmark in a standardized way, as much as possible.

The categorization dimensions impact the basic selection of meaningful benchmark measures as well as the experimentation dimensions. Measures will be specifically addressed in the rest of this deliverable, whereas the other experimentation dimensions (i.e., the workload and the faultload) are addressed in task T23 (i.e., deliverable ETIE3).

3 Systems Studied in DBench

The aim of this section is to give a brief description of the classes of systems that are considered in the framework of DBench and allocate the benchmarking context dimensions, to facilitate definition of pragmatic measures.

To study a representative set of benchmarks, thus allowing for the investigation of a substantial range of benchmark dimensions, we are considering three classes of systems: i) *general purpose operating systems*, ii) *transactional systems* and iii) *embedded systems*.

Operating Systems are critical components of any computer system as they orchestrate the applications that are executed and ensure the interface with the underlying hardware layer. Their malfunctions have a strong impact on the dependability of the global system. It is therefore very important to specifically address their dependability.

Transactional systems constitute the kernel of the information systems used today to support the daily operations of most of businesses. 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, public and government services, etc. These applications comprise the traditional examples of business-critical applications, as well as Web-based applications, which clearly justify the interest for benchmarking the dependability of transactional systems.

Embedded systems are used in several areas, from industrial environments to household utilities and appliances. The so-called next-generation devices increase every day all around us. They tend to be special-purpose devices oriented to specific areas like telecommunication, industrial control, automotive electronics, space applications, medicine engineering, consumer

or office electronics, etc. Complex resources, friendly interfaces, shared information utilities, fast communications, and low cost are new parameters required by users. Many manufacturers have joined this new next-generation embedded systems race flooding the market of processors, controllers, digital signal processors and operating systems. Indeed, the term "embedded" systems is not a precisely defined term. For this work we will use a classical interpretation of embedded systems, in which a system is considered as embedded if it is part of a product whose primary functionality is not computing.

For each class, we expect to explore at least two systems to better identify the main problems related to each class of systems.

- Linux and Windows 2000 provide excellent examples of general purpose Operating Systems (OSs), as they differ significantly in their internal architecture and in their implementation, which constitutes an excellent test with respect to benchmark portability. Also, they differ in their nature, as Linux is an open source Off-the-Shelf (OTS) software while windows is a Commercial OTS (COTS) [1] software.
- OLTP (On-Line Transaction Processing) and Web-based application systems provide two complementary examples of Transactional Systems.
- For embedded systems, we have selected two examples of control systems from different application domains: space and automotive industries.

From *the fault-tolerance* point of view, the OSs have some error detection mechanisms, mainly for detecting some specific faults from the application side and some hardware faults. However, the transactional and embedded systems have internal fault tolerance mechanisms for detecting their own errors (e.g., error detection and recovery mechanisms and integrity enforcing and recovery in transactional systems).

With respect to the **operational environment**, all what is needed to define measures is the maintenance policy, as this information directly impacts the selection of dependability measures. The description of the other environmental conditions is however required for the definition of the workload and faultload (which is out of the aim of this deliverable). Also, note that only maintenance actions requiring human intervention (i.e., achieved manually) are concerned here. Automatic maintenance performed by mechanisms integrated into the system is considered as part of the fault-tolerance mechanisms and is taken into account with the description of the system under benchmark.

After a system failure, either the system can be rebooted to resume its functions (manually or automatically**)** or, if the reboot is not possible, due to system state degradation, human intervention is required for system maintenance. Maintenance can be performed on-line or off-line. We assume that transactional systems have on-line and off-line maintenance possibilities, while embedded systems, due to their nature and to the involved application areas, have only off-line maintenance. With respect to OSs, only reboot actions are considered, although OS reinstallation may be necessary in some particular failure situations.

---

[1] For sake of simplicity, we use the term COTS to designate indifferently Commercial-off-the-Shelf or Off-the-Shelf components.

Figure 2 summarises the three classes of systems considered in DBench.

System nature	Application area	Maintenance
Operating system	General-purpose	Reboot
Transactional system	On-Line Transaction Processing Web-based	Reboot On-line/ off-line maintenance
Embedded system	Automotive control Space control / navigation	Reboot Off-line maintenance

Figure 2: Classes of systems to be studied in DBench

4 Benchmark Measures

We first briefly recall some concepts defined in CF2 that are needed in this deliverable stressing our evolutions since then. Then we put emphasis on measure assessment before concentrating on experimental measures.

Figure 1 shows that dependability benchmark measures can be viewed according to four perspectives:

- The benchmark measure could be qualitative or quantitative. Indeed, when the benchmark scope is to identify system features and possible weaknesses, no numerical values need to be estimated.
- Measures could be related to dependability or dependability and performance (the latter are referred to as performance-related measures).
- With respect to measure extent, we have distinguished 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 individually particular aspects of a system (e.g., fault tolerance capabilities, maintenance and evolution capacities). Each measure characterises one side of the several facets of a system.
- Specific measures could be assessed through experimentation, whereas comprehensive measures could be obtained from experimentation or from experimentation and modelling. For example, the most salient specific measures can be combined into a global model to evaluate appropriate comprehensive dependability measures. Models could be straightforward (combination of specific measures in a simple way) or more complex depending on the system and measure concerned. Indeed, modelling may even consist in the detailed description of the system behaviour, taking into account failure and repair rates of system components as well as interactions between components. Processing of

the model to assess the measures requires the allocation of numerical values to the model parameters. Some of such values may be obtained from experimentation on the system under benchmark (i.e., the specific measures). Other parameter values can be provided from field data or based on past experience related to similar systems (e.g., failure and repair rates). It is worth mentioning that models of large systems made of several components with several dependencies may be very complex . However, for some systems, mainly COTS-based systems for which the details about the system architecture are not known, simple high-level models can be constructed. Such models may require only a reduced effort, thus offering a good trade-off for obtaining comprehensive measures characterizing globally the system with an acceptable effort. Examples of modelling techniques for evaluating comprehensive measures are: block diagrams, fault trees, Markov chains or stochastic Petri nets.

From a practical point of view, the benchmark measures of interest are identified/selected according to the considered system under benchmark and to the benchmark context. The description of the system under benchmark, allocation of the benchmark context dimensions and selection of measures of interest are performed during what is referred to as the analysis step. It is worth mentioning that measures for external use purpose are usually specified in the benchmark. The analysis step identifies those that are meaningful for the particular system under benchmark. However, one might evaluate measures for internal purpose use, in which case such measures are to be defined in the analysis step.

Indeed, the analysis step prepares the experimentation step by defining experimental dimensions. Furthermore, if modelling is required for comprehensive measures, the analysis step usually prepares modelling, by making a deep examination of the system under benchmark.

Experimentation corresponds basically to executing the workload and faultload and observing the system under benchmark (SUB) behaviour.

4.1 Measure Assessment from Modelling and Experimentation

Figure 3 shows the three steps required for obtaining comprehensive measures from modelling and experimental measures (that could be comprehensive or specific) as well as the various links between these steps.

- Link A defines the workload, faultload, experiment configuration required for experimentation and measures together with the associated outputs to be collected on the system under benchmark to assess them.
- Link B provides information for modelling the system under benchmark in order to obtain comprehensive measures from modelling.

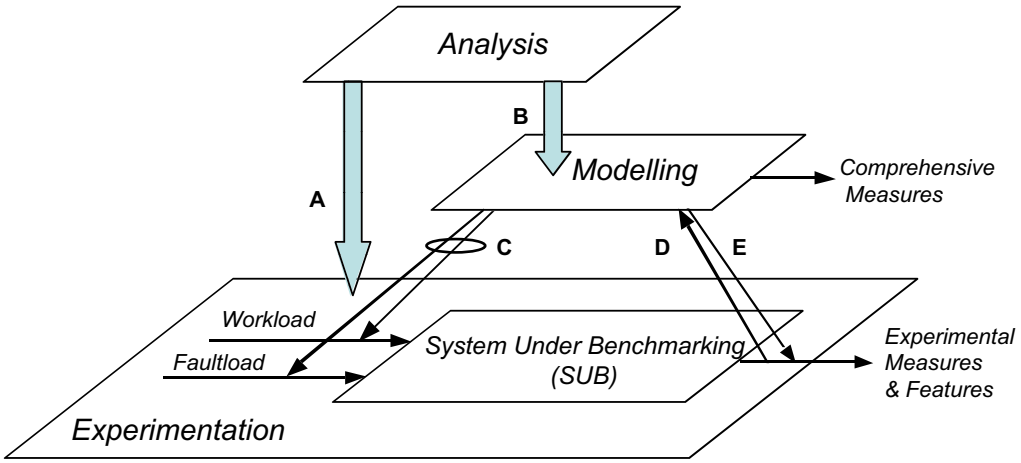


Figure 3 - Dependability benchmarking steps

Usually experimentation and modelling are used in a complementary way to assess system dependability [Arlat *et al.* 2001]. The links shown in Figure 4 correspond to the following:

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

In the next section, we give emphasis to experimental features and measures.

4.2 Experimental Measures

Experimentation is tightly connected to the system on which the experiments will be performed (i.e., it closely depends on its actual implementation, including its accessibility and observability points). On the other hand, the analysis step, taking into account the system under benchmark nature, application area and the operational environment at a high level (i.e., a class of systems defined in a generic way), can only define the workload and faultload at a high level (i.e., generic workload and faultload) and specify the outputs required from the experiments (and not the precise measurements) to assess the benchmark measures. Also, the analysis step should define the "outside" configuration required for the experiments in order to "simulate" the operational environment of the system under benchmark. Indeed, for being able to carry out controlled experiments, experimentation should embody supporting facilities. In practice, the simulation of the outside configuration could be integrated in the workload or provided separately. More generally, a generic benchmark can only specify at a

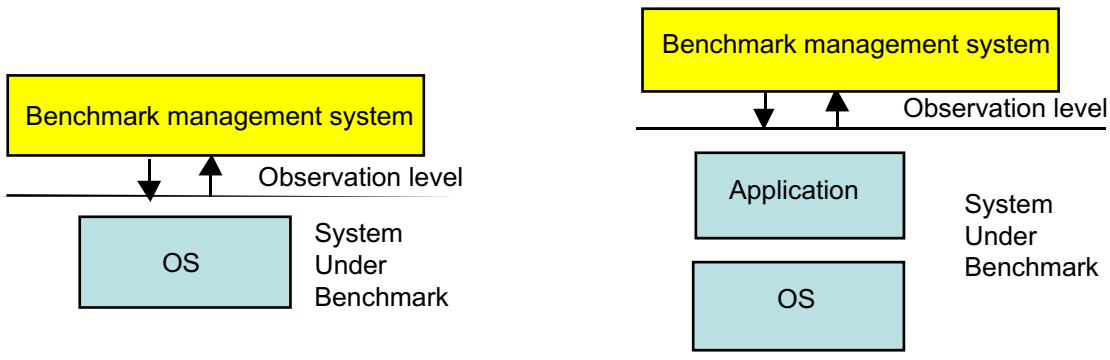
high-level the workload and faultload, the kind of outputs required from the experiments to assess the benchmark measures, as well as the configuration under which the experiments should be performed (the latter is referred to as experimental configuration).

This means that the generic workload and faultload, the required outputs and the experiment configuration, defined at a high level during the analysis step or by a benchmark specification, should be refined, in order to be implemented at the SUB level. This refinement should be accomplished according to procedures and rules included in the benchmark. In particular, these procedures and rules should address scalability of the benchmark and configuration disclosures.

To perform experiments, an external host system is usually required. This host system is part of the **benchmark management system**. Its role is to supervise the experiments. It activates the SUB according to the workload and faultload and collects the results.

5 Benchmark Measures for the Systems Studied in DBench

The system outputs delivered to the benchmark management system are processed to assess measures of interest. Data items are collected at the interface between the SUB and the benchmark management system. This interface determines the observation level. If the benchmarked system is an OS the observation level corresponds to the OS output level, if the SUB is a transactional or an embedded system, the observation level corresponds to the output of the system. However, for system validation and for identifying weak parts of transactional or embedded systems, lower level observation points may be required (e.g., observation at the OS outputs to better analyse possible problems). Figure 4 examples of experimental measures and observation level, for an OS and for an embedded system running of top of an OS.



a) OS benchmark

b) Embedded system benchmark

Figure 4 – Example of experimental measures and observation level

As stated earlier, measures of interest are directly linked to the life cycle phase in which the benchmark is performed. In addition, for both transactional and embedded systems, results

that may be obtained from early and late integration steps usually differ, as the benchmark scopes are usually unlike.

During the last integration steps, the benchmark is generally used to characterise the final product (including the hardware platform, the OS and the application software). Of course the results will be more accurate and the end of the integration phase, when the system is almost ready for shipment. In which case, the benchmark results could be provided to the future system buyer to increase their confidence in the system they are acquiring. Indeed system vendors can display the system dependability benchmark results in the same manner as they exhibit its functional characteristics.

As for performance benchmark, we limit ourselves to a small number of measures that could be evaluated in a standard way. Also, we favour service-oriented measures, i.e., comprehensive measures that reflect the end-user point of view and consider both dependability and performance-related measures.

The final product could be characterised directly from experimentation, assessing comprehensive measures such as the number of transactions executed in presence of faults or system throughput in presence of faults, or system failure modes. However, these experimental measures may be complemented by comprehensive measures evaluated through modelling. As transactional systems are usually repairable, system availability is a meaningful measure. However, embedded systems and particularly the automotive and space control applications considered in DBench, the systems are not repairable online, thus safety and reliability are more appropriate measures. The parameters of the models can be obtained from the benchmark results.

During the early system integration steps, the benchmark can be used as a validation means to i) identify possible weak parts in the system architecture and ii) assess the fault tolerance mechanisms of the system. The ultimate aim is to improve the architecture and fault tolerance capabilities, if necessary. Also, the benchmark could be used to compare and select particular components of the system being integrated (the underlying platform, the database engine, the OS, etc.). In addition, for some components, the system integrator can make use of benchmark results made available by other organisations.

Usually, the results are mainly intended for internal purpose use. Thus the main users of the benchmark results are the system integrators, as they know very well the details of the system being integrated as well as the conditions under which the benchmark is performed. However, as stated earlier, in the framework of system benchmarking, even without using necessarily a standard way, it is better to define consistent methods for assessing fault tolerance specific measures. This is the objective of Section 6.

It is worth mentioning that, the benchmark approach applies to products available on the market, as well. This situation is very similar to what is described for the last integration steps described above..

In the rest of this section we consider successively operating, transactional and embedded system measures of interest.

5.1 General Purpose Operating Systems

The main scopes of OS benchmarks are twofold: i) explore its dependability features and measures and ii) compare alternative candidates, to identify a (best) potential OS candidate to use into a design. The two scopes are not totally independent, as the features can be compared for alternative candidates. As a consequence, in this section we will mainly concentrate on the benchmarking of an OS to explore its somewhat generic dependability features and measures.

It is worth mentioning that given the generic dependability features of general purpose OSs, it is possible that the benchmark be conducted by third-party entities more or less independent from the system integrator for whom they are performing the benchmark. On one hand, one may think of a consulting agency, on the other hand, a typical example would be a specific department of a large company.

Measures of Interest

An OS interacts directly with the application or middleware layer and the hardware layer. One of the dependability features of an OS could be to "resist" faults induced by the two layers it interfaces, i.e., the so-called **robustness feature**. These features are measured through the observation of the OS behaviour following fault activation.

In addition to robustness that will be detailed later, several other features are of interest. Some examples are listed hereafter.

Some hardware error detection and coverage features For instance, the OS might include mechanisms for preservation of the system file integrity in case of a hardware error, and for file corruption diagnosis or auto-repair, in case of software failures. Such features could be very important for some application areas. However, generally, such facilities are more or less part of the middleware, rather than the OS per se.

Error confinement feature An OS may have the possibility to avoid error propagation from one application to another one that does not communicate with it, but the two applications run on the same OS. For example if the OS does not include efficient memory protection mechanisms, an application (that is not completely validated) might corrupt the memory space allocated to another application, leading to the failure of the latter. The failure mode of the affected application depends of the memory segment corrupted: code or data segment. Thus in some cases, an error code is returned, but in some other cases, the application failure consists in providing erroneous outputs, which is much more critical and more difficult to identify in absence of an *oracle* (i.e., the *a priori* knowledge of the behaviour of the target application without fault).

Time measurements could be of interest as well. For instance, OS-related real-time measures such the OS reboot and reset times or task execution time and switching time should be important for real-time applications (more for control applications than for transactional applications). The reboot time could be deterministic when carried out under normal conditions. However, it could be longer and non-deterministic when performed from failure states (after a kernel hang, a crash or even a power supply failure). In the latter case, the reboot time depends on the checks and corrections performed by the OS to restore for example the OS integrity. When these times are non deterministic, a time interval should be provided rather than a single value (or the average and variance).

In the rest of this section, we put emphasis on robustness features, before giving some concluding comments.

Robustness Features

Let us consider the case where one or more applications are executed on top of the OS under benchmarking. Following fault activation (simulating an erroneous input or a hardware fault), the consequences can be observed. The various situations and how to observe them are detailed here after.

i) The kernel crashes. A crash is easily observed as the OS issues some messages. Also, these messages can be directed towards a log file to be analysed. The analysis may reveal in some cases the part of the OS that is affected.

ii) The kernel hangs. A hang is a non-observable internal state. Nothing is communicated outside the OS. This state can be made detectable by the benchmark management system (by sending a "ping" signal, for example). Also, benchmark management system can initiate an automatic reboot of the OS.

iii) Following a kernel call, the OS returns an error code to the application or, in the context of a benchmark, to the benchmark management system. Examples of error codes are "Inexistent file" or "Out of memory". The error status is an event internal to the application that should handle it. Thus, it is possible to make it available to the benchmark management system. These error statuses play an important role for error diagnosis and recovery. For example, it is preferable to have an error code indicating that a file is corrupted to an "undefined error" (the latter is a particular error status that could be returned). In the first case, the restoration of the indicated file can avoid system down time and thus does not affect its availability. A kernel that returns very few "undefined error" is preferable to a kernel that returns most of the time "Undefined error". Hindering (i.e., the error code returned is not correct) is an interesting facet of the behaviour. However, the identification of such a situation requires deep analyses. This can only be envisaged when faults are injected in the parameters of the kernel call at the application level. In addition, one has to determine for each kernel call the various error status associated with each affected parameter.

iv) The kernel issues an exception that can be handled by the application, as in the case of an error status. Some exceptions cannot be handled by the application itself as the application is aborted (or killed). This is the case for example of "Bus error" or "Address error". The consequences of the latter exceptions are usually observable externally to the application.

v) A result is returned to the application. In the context of a benchmark, it is returned to the benchmark management system. This result might be correct or incorrect. As a consequence, the application might fail or hang. Analysis of incorrect results can only be possible if an oracle is available.

Concluding Comments

In practice, both the characterisation and the distribution of the events and states presented above are of interest.

Also, it is important to note that the outcomes are not exclusive (several of these events can occur as the result of the application of a specific faultload). Thus, related timing observations are also of interest. For example, the observation of an *exception* before a *kernel hang* is preferred to no exception at all. Although precise timing measurements are not easy to achieve especially in the case of a COTS (except for the case of exception or error codes returned to the application), ordering of events can be much easier to achieve.

Benchmarking an OS consists in identifying specific features and assess some measures such as those given above. However, i) some of them may be important for all application areas (e.g., robustness), ii) others may have to be adapted or can be specified more precisely according to the application special requirements and iii) not all of the measures are necessary and even meaningful for all applications. For example, for transactional systems, real-time related measures could be useless and for embedded systems with a critical application running alone on top of the OS, the error confinement property is meaningless. Moreover, embedded systems usually use reduced versions of the OS, as some functionalities, intended for general purpose use, are not used. The benchmark can thus better target the part that is used for the embedded system.

Finally, all what precedes concerns the benchmarking of the OS itself and the measures of interest are those presented in this section or similar ones. These measures are intended to characterise the OS behaviour directly as indicated in Figure 5a. Comparing two OSs consists in benchmarking each of them and comparing the OS benchmark measures. Nevertheless, if the application software is available, the OS and the application can be benchmarked together as indicated in Figure 5b in which the workload triggers the application that, in turn, makes system calls to the OS. In this case, the measures that are assessed are intended to characterise the application together with the OS as we will see in the next section for transactional and embedded systems. Thus comparing two OSs may consist in benchmarking the application system running alternatively the compared OSs to compare their impact on the whole system behaviour, rather than comparing directly the two OS benchmark results.

5.2 Transactional Systems

Given the huge variety of transactional system configurations used in practice to run transactional applications, the definition of the system under benchmark is tied to the transactional workload instead of being defined in a structural way. In other words, the system under benchmark can be any system able to run the workload according to the conditions specified by the dependability benchmark.

A typical computer transaction includes normally reading from, writing to, or updating a database system for things such as inventory control (goods), airline reservations (services), or banking (money). At the database level, a database transaction is a set of database operations that comply with the well-known ACID properties: Atomicity, Consistency,

Isolation, and Durability. A business transaction is comprised of one or more database transactions.

Measures of Interest

We consider two sets of measures i) comprehensive measures for external purpose use and ii) some specific measures, mainly intended for internal purpose use. Typical measures in the two sets are successively addressed hereafter.

External purpose use

In presence of faults, either the fault is tolerated and the service is delivered with possible degradation, or some transactions are lost or the system fails. Due to the many error detection and fault tolerance mechanisms included in a transactional system at various levels, several degraded service states or, equivalently, system failure modes may take place. These failure modes are directly related to the mechanisms implemented at the various places (hardware and OS levels, database engine level and at the application level such as workload consistency tests). Knowledge of the failure modes (nature and distribution) gives an accurate representation of the system behaviour in presence of faults.

These situations can be characterised or assessed by sets of complementary measures[2] that are not totally independent. We give examples of four sets, that i) performance in presence of the faultload, ii) number of transactions aborted due the faultload and iii) system availability (or unavailability) due the faultload and iv) failure modes under the benchmark faultload. They are defined as follows:

- **Number of transactions per minute in presence of the benchmark faultload** It measures the impact of faults on the performance and favours systems with higher capability of tolerating faults, fast recovery time, etc.
- **Number of transactions aborted because of the benchmark faultload** – 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.
- **Availability under the benchmark faultload**, during the benchmark period – It measures system availability during the benchmarking period. It is given by the ratio of total time the system is unavailable over the total benchmark time.
- **Failure modes under the benchmark faultload –** Gives the impact of the faultload on service delivery, as defined from the end-user point of view.

Also, additional measures related to price per transactions, inspired from performance benchmarks, could be assessed.

For assessing these measures, we presume the following experimental conditions. After fault activation, the system may continue executing transactions normally if nothing wrong is detected. Otherwise it starts a recovery procedure, depending on the fault effects and the

---

2 Some examples of possible measures have already been given in CF2 (Section 5.1.2).

error detection and diagnosis mechanisms in the system, or may need to be restarted completely if recovery is not enough. Also, we assume that the complete restart of the system (i.e., the OS and the database) will be performed automatically by the benchmark management system, whenever it detects a system crash.

All these measures can be obtained by the same set of experiments: execute the benchmark workload in the presence of the faultload and collect the required outputs. We have decided to adopt the workload of the well-established TPC Benchmark™ C (TPC-C). This workload emulates all the terminals and their emulated users during the benchmark run. TPC-C provides the number of transactions executed per minute that 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[3]. When executed under the benchmark faultload conditions, it will provide the number of transactions per minute in presence of faults.

The benchmark management system not only simulates the database clients and submits the transaction profile to the system under benchmark but also records all the information needed to obtain the measures. Thus all the necessary information for assessing the measures for external purpose use should be obtained from the benchmark management system.

With respect to the time required to perform the benchmark for obtaining system availability under workload, at this stage of the project, we think we have to define a minimum and a maximum time. The actual benchmarking time must be measured (by the benchmark management system and must be in the interval specified (i.e., between a minimum and a maximum time).

Internal purpose use

Specific dependability measures such as error detection efficiency, globally or at a given level, can be easily included in a benchmark for transactional systems. Indeed, a given system component can be characterised according to several specific dependability measures. For example, specific dependability measures for the database engine are error detection efficiency, data recovery efficiency, data integrity violations and violations of transaction properties due to faults.

As they are intended for internal purpose use, the system integrators have some autonomy for implementing inner observation points and collecting the information. Section 6 will give example of result outputs and how to present them.

5.3 Embedded Systems

Difficulties arise when selecting the best option for a new product. System specifications help to select a particular host platform. However, dependability is highly important for selecting the most appropriate hardware and software components.

---

3 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.

Measures of Interest

The basic attribute of any application, transactional or embedded, is delivery of specified services. Thus, the key dependability benchmarking measure of "performance in the presence of faults" will indeed be the key attribute to measure in a first step. It is worth mentioning that the comprehensive dependability measures are defined with respect to the main services of the application.

External purpose use

For the same reasons as for transactional systems, we present an initial limited set of measures that we believe to be relevant for a large number of embedded systems. Other measures such as jitter and

- **Response time in presence of faults** – The response time of the embedded application, especially for a real-time application, is very important. Assuming that the timing characteristics of the application are known, the behaviour in the presence of faults can be assessed to characterise "performance" degradation in presence of faults.
- **Throughput in presence of faults** – The throughput of the system can be of high importance.
- **Stability** – The stability of the system is a measure of how the failure of one or more components affects the overall behaviour of the system. Incorrect or incomplete results may arise. Their impact on the services provided can be studied by observing the various failure modes of the system.
- **Failure modes under the benchmark faultload -** Incorrect or incomplete results may arise. Their impact on the service of the application must be studied through the analysis of the various failure modes of the application.

The first two measures require only time measurement and can be obtained easily by including such timing functions in the benchmark management system.

The last two measures do not require inside access to the system: the system outputs and their comparison with outputs obtained under the same conditions, without faultload, allow identification of the system behaviour in presence of faults. However, if more detailed information concerning the internal behaviour of the system is needed, this may require the instrumentation of the system.

Internal purpose use

Generally, several error detection mechanisms are implemented in the system, to detect for example: invalid address accesses, instruction protected error, data protected error, bus error, floating point unavailability, etc. as for transactional systems, the effectiveness of these mechanisms can be assessed experimentally. Fault tolerance mechanisms implemented at the system level can also be assessed.

In practice, the system integrator identifies the most salient dependability features (either from system analysis or, better, from system dependability evaluation based on system

modelling). These specific measures are then assessed experimentally. Section 6 gives recommendations on how to assess them in a consistent way.

6 Measures for Systems Implementing Fault Tolerance Mechanisms

In the previous section, the fault tolerance measures have been put in the category of specific measures for internal purpose use benchmarks. This means that they do not require to be necessarily assessed in a standard way. The main reason is that, usually fault tolerance mechanisms heavily depend on the system in which they are implemented. Besides, their assessment may require the system under benchmark instrumentation, which is not in kept with standard benchmarking rules. In addition, they do not necessarily interest the end-users.

Usually, a system has several mechanisms tolerating different fault types and implemented at different levels. These mechanisms are complementary and what is important, from the end-user point of view, is the overall behaviour of the system, as resulting from all of them. For instance, it is not meaningful to compare systems based only on measures related to their fault tolerance mechanisms in isolation, as the impact of these mechanisms may be composed to form the overall behaviour, even though it is important to identify which kinds of faults they can tolerate. Indeed, the efficiency of the whole fault tolerance mechanisms directly impacts the comprehensive measures assessed by the benchmark for external purpose use. For example, measures such as system throughput or the number of transactions executed in presence of faultload depend on the combined efficiencies of all system fault tolerance mechanisms.

Nevertheless, during system development/integration, it is recommended to assess the efficiencies of the various fault tolerance mechanisms, because i) it is still time to improve them, if required, and ii) their (good / outstanding) values can be an excellent selling point when displayed by the system vendors. However, without an accurate description of the meaning of the displayed figures along with the conditions under which they have been obtained, these figures are useless.

Also, the accurate knowledge of the efficiency of fault tolerance mechanisms is supportive for evaluating comprehensive measures, based on modelling. A crucial problem concerns the matching between the parameters used in the model, for which results are expected from experimentation, and what is really assessed by the experiments. The models make some assumptions that could be different from the experimental conditions. It is thus very important that the experimental conditions as well as the assessment method be specified clearly, to incorporate the right numerical values in the models.

To sum up, even though specific fault tolerance measures cannot be part of a standard benchmark, due to the diversity of fault tolerance mechanisms that could be implemented, it is recommended to present the results obtained for internal purpose use in a form that can make them exploitable by other entities.

The aim of this section is to give some hints regarding the way experimental outputs associated with specific fault tolerance specific measure could be presented to make them available to others. We will first discuss the possible outcomes of experiments intended for

evaluating fault tolerance specific measures, then we will give examples of various coverage factors that could be derived from these experiments.

6.1 Experimental Conditions

Let us recall that a fault tolerance mechanism is defined by its efficiency, i.e., the coverage factor and its latency (or duration). A high coverage factor may involve a high latency and usually, a tradeoff is needed. Although measuring latency is not an easy problem, we would like to put emphasis on the evaluation of the coverage factors. Also, our aim is to be as generic as possible in order to give guidelines rather than address a specific measure.

We assume the following experimental conditions. A fault is injected in the system and the miscellaneous observation means allow to check if it has been activated or not. When a fault is activated, assuming that error detection mechanisms are implemented in the system, the error is either detected or the system fails or nothing is observed during the experiment duration. If the system implements error recovery mechanisms the detected error can be correctly covered or the system fails (i.e., the error is non-confined). For each injected fault, a series of events occur and the system end-up in one of these states. This is presented in Figure 5 that gives an example of experiment outcomes. Indeed this figure includes a strange situation that could be observed from experiments: the system fails (state ND) before an error is detected (arc from ND to D). Such situation is non-expected. Indeed, the error may have propagated to another error that has been detected by other fault tolerance mechanisms implemented in the system. This is just an example of uncommon situations that have indeed been observed on a real set of experiments. However, other singular situations may be identified by the experiments.

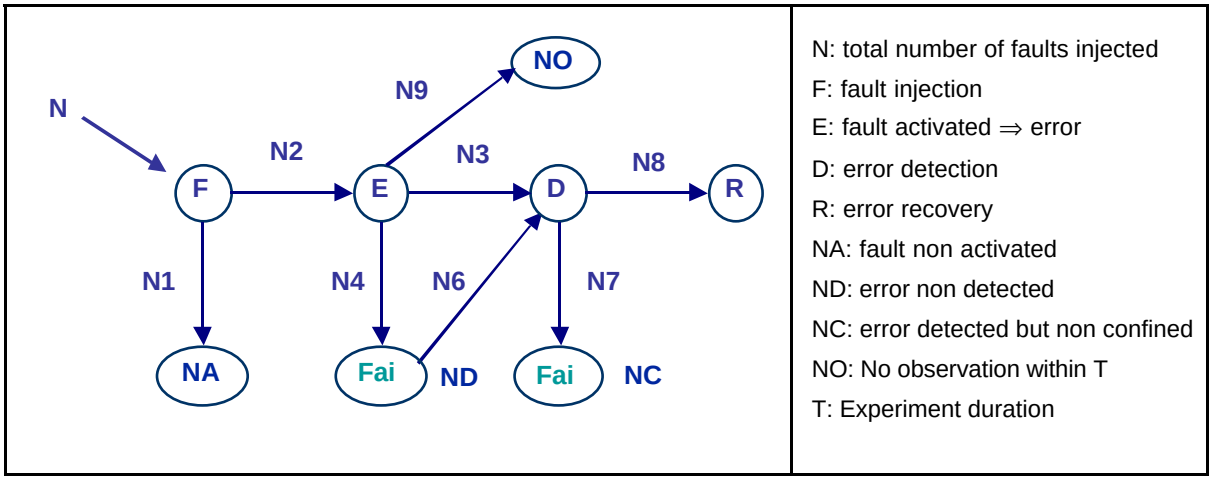


Figure 5 – Possible experimentation outputs

In Figure 5, an Ni (associated to the arc between two states A and B) gives the cumulative number of times the system entered state B coming from state A. In addition the time elapsed from fault injection to event observation should be recorded to assess the latency associated to these events.

A set of experiments may be performed for each class of faults considered in the benchmark or, for all classes of faults, depending on the aim of the assessment. A separate assessment allows more accurate knowledge and may require more observation points. It could help during the early development steps. On the other hand, an overall assessment provides a global vision and may be preferable in the later integration steps.

6.2 Coverage Evaluation

Several factors can be evaluated from the graph given in Figure 5. They are defined as follows:

- Fault activation factor: N2/ N
- error non confinement factor: (N4 + N7) / N2

Concerning error detection and recovery factor, several possibilities are envisaged:

- Error detection factor:

  State D following an error detection is entered either from state E or state ND, in addition, the detection factor could be evaluated either with respect to the number of activated faults (N2) or with respect to the total number of injected faults. This leads to the four subsequent possibilities.

  N3 / N2

  or N3 / N

  or (N3 + N6) / N2

  or (N3 + N6) / N

- Error recovery factor:

  State R following an error recovery is entered from state D that is entered either from E or ND. The coverage factor could be evaluated either with respect to the total number of injected faults or with respect to the number of activated faults (N2) or the number of detected errors (N3 + N6). This leads to the three subsequent possibilities.

  N8 / N

  or N8 / N2

  or N8 / (N3 + N6)

None of these error detection and recovery factors is better or more accurate or more convenient, than the others. However, if the delivered number is not clearly defined, it could be used in a wrong manner. That is why it is very important to specify clearly the conditions under which they have been obtained. Indeed, when presenting the experiment results it is recommended to provide rough results and it is up to the users to evaluate what they need. Better, the measures could be presented in a graphical manner or in the form of a matrix, to give a global view of the various outputs, but this is not mandatory.

7 Summary and Conclusion

This deliverable was mainly devoted to benchmark measures and measurements to be performed on the system under benchmark to obtain them. Measurement is used in a broad sense, it refers to the information to be collected on the system under benchmark to obtain the measures of interest.

Before concentrating on measures, we gave a summary view of the benchmark dimensions. Indeed, our vision and classification to these dimensions have slightly evolved since our previous deliverable CF2. We put emphasis on how these dimensions impact the selection of measures of interest. The three groups of dimensions are: categorisation, measure and experimentation. Measures of interest for a benchmark depend on the categorisation dimensions and, in turn, experimentation dimensions depend on the retained measures.

The categorisation group is composed of two sets of dimensions: those describing the system under benchmark and those giving the benchmark context. All dimensions are determinant for identifying the measures of interest. One of the benchmark context dimensions is use purpose. Referring to performance benchmarks that are defined as standards, we distinguished two benchmark use purposes: external and internal.

Measures for external purpose use should be defined in a standard way together with the experimental conditions allowing their assessment. They should be made publicly available. As a consequence, they require only information items that are available from outside the system under benchmark or that can be obtained easily from the benchmark management system. The standard conditions will be defined in WP3 of DBench.

Measures for internal purpose use, do not require necessarily standard conditions to be obtained and may require inner information. Usually, they are evaluated by system integrators for system validation. Indeed, at this stage of the project, we are not aiming at defining them (as well as the conditions needed to evaluate them) in a standard way, because of the large variety of fault tolerance mechanisms that could be implemented on a real-life system. However, as most of these measures address specific fault tolerance features of the system, it is recommended to assess them in a unified manner. The last section of this deliverable was devoted to recommendations for performing the experiments and presenting the results in a way that allows other entities to use them.

Measures could be either qualitative or quantitative, related to dependability or performance and dependability, comprehensive or specific and assessed from experimentation or modelling and experimentation. Given the aim of this deliverable, we put emphasis on experimental measures.

To illustrate, the previous concepts, we presented, according to the categorisation dimensions, the three classes of systems that will be addressed in DBench, namely general purpose operating systems, transactional systems and embedded system. Then we presented for each of them examples of measures that can be evaluated according mainly to the benchmark scope and to the use purpose.

Operating systems can be benchmarked i) in a generic way without any particular application or ii) using a particular application as a special workload. In the first case, only generic

features can be assessed, such as robustness with respect to hardware and application faults. When using a particular application as a workload, results are more accurate and can target only features of particular interest to the application.

Concerning transactional systems and embedded systems, for each of them, we have presented examples of measures, for external and internal purpose use respectively. For external purpose use, a limited number of dependability and performance-related measures intended to characterise their behaviour in presence of faults have been defined. All these measures are comprehensive measures (defined with respect to the system service as perceived by the end-users). They include the number of transactions per minute and system availability in the presence of faultload, for transactional systems, and system throughput in presence of the faultload and system response time in presence of faults for embedded systems.

Measures for internal purpose use mainly concern the various fault tolerance mechanisms implemented in the system.

For each class, two systems will be considered to allow validation of the concepts and make some cross exploitation of results possible.

8 References

[CF1] J. Arlat, K. Kanoun, H. Madeira, J. V. Busquets, T. Jarboui, A. Johansson and R. Lindström, State of the Art, Available at http://www.laas.fr/dbench/delivrables.html, DBench Project IST 2000-25425 Deliverable, N°CF1, September 2001 (Also LAAS Report 01-605).

[CF2] H. Madeira, K. Kanoun, J. Arlat, Y. Crouzet, A. Johanson and R. Lindström, Preliminary Dependability Benchmark Framework, Available at http://www.laas.fr/dbench/delivrables.html, DBench Project IST 2000-25425 Deliverable, N°CF2, September 2001 (Also LAAS Report 01-604).

[BDEV1] "*Dependability Benchmark Definition: DBench Prototypes*", DBench Project IST 2000-25425, June 2002.

[ETIE2] "*Fault Representativeness*", DBench Project, IST 2000-25425, June 2002

[ETIE3] "*Workload and Faultload Selection*", DBench Project, IST 2000-25425, June 2002.