



DEPARTAMENTO DE
ENGENHARIA
INFORMÁTICA

A Dependability Benchmark for OLTP Application Environments

Marco Vieira

Henrique Madeira

Título: Relatórios Internos do DEI

ISSN: 0873-9293

Capa: Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Edição: Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

© 2003 Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

A Dependability Benchmark for OLTP Application Environments

Marco Vieira

ISEC/CISUC - Polytechnic Institute of Coimbra
3031 Coimbra
Portugal
mvieira@isec.pt

Henrique Madeira

DEI/CISUC - University of Coimbra
3030 Coimbra
Portugal
henrique@dei.uc.pt

Abstract

The ascendance of networked information in our economy and daily lives has increased the awareness of the importance of dependability features. OLTP (On-Line Transaction Processing) systems constitute the kernel of the information systems used today to support the daily operations of most of the business. Although these systems comprise the best examples of complex business-critical systems, no practical way has been proposed so far to characterize the impact of faults in such systems or to compare alternative solutions concerning dependability features. This paper proposes a dependability benchmark for OLTP systems. This dependability benchmark uses the workload of the TPC-C performance benchmark and specifies the measures and all the steps required to evaluate both the performance and key dependability features of OLTP systems, with emphasis on availability. This dependability benchmark is presented through a concrete example of benchmarking the performance and dependability of several different transactional systems configurations. The effort required to run the dependability benchmark is also discussed in detail.

1. Introduction

This paper proposes a dependability benchmark for On-Line Transaction Processing (OLTP) systems. The goal of this benchmark is to provide a practical way to measure both performance and dependability features (with emphasis on availability) of OLTP systems. These systems constitute the kernel of the information systems used today to support the daily operations of most of the businesses and comprise some of the best examples of

business-critical applications. However, in spite of the pertinence of having dependability benchmarks for this important class of systems, the reality is that no dependability benchmark (or even a really practical approach to characterize dependability features such as availability) has been proposed so far, in a clear contrast with the benchmark of performance, where the Processing Performance Council (TPC) organization has started long ago a family of performance benchmarks.

Database Management Systems (DBMS), which are the kernel of OLTP systems, have a long tradition in high dependability, particularly in what concerns to data integrity and recovery aspects. However, in most of the DBMS the effectiveness of the recovery mechanisms is very dependent on the actual configuration chosen by the database administrator. The problem is complex, as tuning a large database is a very difficult task and database administrators tend to concentrate on performance tuning, often disregarding the recovery mechanisms. The constant demands for increased performance from the end-users and the fact that database administrators seldom have feedback on how good a given configuration is concerning recovery (because faults are relatively rare events) largely explain the present scenario.

Performance benchmarks such as the TPC benchmarks have contributed to improve peak performance of successive generations of DBMS, but in many cases the systems and configurations used to achieve the best performance are very far from the systems that are actually used in practice. The fact that many businesses require very high availability for their database servers (including small servers used in many e-commerce applications) shows that it is necessary to shift the focus from measuring pure performance to the measurement of both performance and recoverability. This is just the goal of the dependability benchmark proposed in this paper.

Funding for this paper was provided, in part, by Portuguese Government/European Union through R&D Unit 326/94 (CISUC) and by DBench Project, IST 2000 - 25425 DBENCH, funded by the European Union.

A version of this report has been submitted for publication outside of Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra (DEI-FCTUC) and it will be copyrighted if accepted. The distribution of this report outside DEI-FCTUC is aimed at early dissemination of its content and is restricted to peer dissemination and specific requests.

Computer benchmarking is primarily an experimental approach. 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. The first aspect (ability to reproduce) gives confidence in the benchmark results and the second (ability to generalize) makes the benchmark results meaningful and useful beyond the specific set up used in the benchmark process.

In practice, benchmark results are normally reproducible in a statistical basis. On the other hand, the necessary generalization of the results is inherently related to the representativeness of the benchmark experiments. The notion of representativeness is manifold and touches almost all the aspects of benchmarking, as it really means that the conditions used to obtain the 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 is in fact the key that turns a benchmark into a standard. In other words, a 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 (standard) way of measuring specific features of a computer system and, consequently, a way to compare different systems.

The concept of benchmarking can then be summarized 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 characterize the computer features that help the vendors/integrators to improve their products and help the users in their purchase decisions.

A dependability benchmark can then be defined as a specification of a standard procedure to assess dependability related measures of a computer system or computer component. Given the agreement nature of all benchmarks, it is obvious that a real dependability benchmark can only be established by the computer industry or by the user community. Nevertheless, the dependability benchmark proposed in this paper shows a possible path to benchmark the dependability of OLTP systems and contributes this way to the technical discussion required to create the conditions for the establishment of actual dependability benchmarks.

The paper is structured into six sections. Section 2 presents the key components and the main properties of a dependability benchmark for OLTP systems. Section 3

presents the DBench-OLTP dependability benchmark, discussing and justifying at the same time the most relevant design choices. Section 4 presents several benchmarking examples that have resulted from an extensive study meant to validate the key properties of the proposed dependability benchmark. Section 5 discusses the effort required to run this benchmark and section 6 concludes the paper.

2. Benchmarking dependability in OLTP application environments

A typical OLTP environment consists of a number of users managing their transactions via a terminal or a desktop computer connected to a database management system (DBMS) via a local area network or through the Web. An OLTP system is thus a typically client-server system or a multi-tier system. In a simplified approach, the server is composed by three main components: the hardware platform (including the disk subsystem), the operating system, and the transactional engine. Most of the transactional systems available today use a DBMS as transactional engine, which is in practice the main component of any OLTP systems, assuring not only the transactional properties but also the recovery mechanisms.

Dependability is an integrative concept that includes the following attributes [Laprie 1995, Avizienis 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.

Among all the possible dependability attributes, availability is one of the most relevant in databases and OLTP systems in general. Thus, the proposed benchmark is particularly focused on the availability of the system under benchmark.

The main problem in measuring the availability of a given computer system or component is that this measure is very dependent on the fault probability, which is dependent on many factors, either internal to the system (hardware and software) or external (environment or human made). Assessing system availability is in fact a very difficult problem and has been addressed in the dependability research community by using both model-based and measurement-based techniques. The former include analytical [Trivedi 94] and simulation [Jenn 95] techniques and the latter include field measurement [Gray 90], fault injection [Hsueh 97, Carreira 98] and robustness testing [Siewiorek 93, Koopman 99].

Our proposal to dependability benchmarking is mainly inspired on measurement-based techniques. Comparing to well-established performance benchmarks, this new type of benchmarks includes two new elements: 1) the **measures related to dependability** and 2) the **faultload**. In this way, the main components of the proposed dependability benchmark are:

- *Workload*: represents the work that the system must perform during the benchmark run.
- *Faultload*: represents a set of faults and stressful conditions that emulate real faults experienced by OLTP systems in the field.
- *Measures*: characterize the performance and dependability of the system under benchmark in the presence of the faultload when executing the workload. The measures must be easy to understand and must allow the comparison between different systems.
- *Benchmark procedure and rules*: description of the procedures and rules that must be followed during a benchmark run.
- *Experimental setup*: describes the setup required to run the benchmark.

In addition to the repeatability and representativeness properties mentioned before, benchmark portability [Gray 93] is also a very important property. In the DBench-OLTP the faultload is clearly the most problematic component concerning portability, as it is necessary to assure that the faults used in the faultload are equivalent across the different target systems (see section 3.4).

3. DBench-OLTP: Component design and properties discussion

The DBench-OLTP dependability benchmark uses the basic setup, the workload, and the performance measures specified in the TPC-C and introduces the new components mentioned before: measures related to dependability and faultload. This section presents and discusses the DBench-OLTP dependability benchmark, with particular emphasis on the new components.

3.1 Experimental setup and benchmark procedure

Figure 1 presents the key elements of the experimental setup required to run the DBench-OLTP. As in TPC-C, the main elements are the System Under Test (SUT) and the Driver System.

The goal of the driver system is to emulate the client applications and respective users and control all the aspects of the benchmark run. In DBench-OLTP the driver system has been extended to handle the insertion of the faultload. Additionally, the driver system also records the raw data needed to calculate the benchmark measures (measures are computed afterwards by analyzing the information collected during the benchmark run).

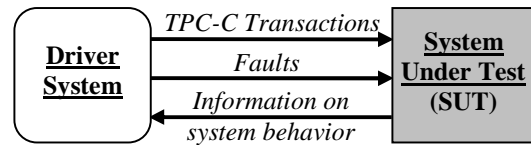


Figure 1 – DBench-OLTP experimental setup.

The SUT represents a client-server system fully configured to run the workload, and whose performance and dependability is to be evaluated. From the benchmark point of view, the SUT is the set of processing units used to run the workload and to store all the data processed. That is, given the huge variety of systems and configurations used in practice to run OLTP applications, the definition of the SUT is tied to the transactional workload instead of being defined in a structural way. In other words, the SUT can be any (hardware + software) system able to run the workload under the conditions specified by the benchmark.

A DBench-OLTP run includes two main phases:

- *Phase 1*: First run of the workload without any (artificial) faults. This phase corresponds to a TPC-C measurement interval (see [TPC-C 02]), and follows the requirements specified in the TPC-C standard specification. Phase 1 is used to collect the baseline performance measures that represent the performance of the system with normal optimization settings. The idea is to use them in conjunction to the other measures (see below) to characterize both the system performance and dependability.
- *Phase 2*: In this phase the workload is run in the presence of the faultload to measure the impact of faults on the transaction execution (to evaluate specific aspects of the target system dependability). As shown in Figure 2, Phase 2 is composed by several independent injection slots. An injection slot is a measurement interval during which the workload is run and one fault from the faultload is injected.

In order to assure that each injection slot portrays a realistic scenario as much as possible, and at the same time assures that important properties such result repeatability and representativeness of results are met by the DBench OLTP dependability benchmark, the definition of the profile of the injection slot has to follow several rules. The following points briefly summarize those rules (see also figure 2):

- 1) The SUT state must be explicitly restored in the beginning of each injection slot and the effects of the faults do not accumulate across different slots.
- 2) The tests are conducted with the system in a steady state condition, which represents the state in which the system is able to maintain its maximum transaction processing throughput. The system achieves a steady state condition after a given time executing transactions (steady state time).

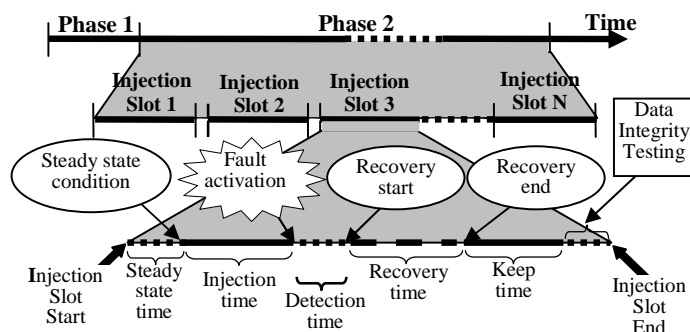


Figure 2 – Benchmark run and injection slots.

- 3) Each fault is injected a certain amount of time (injection time) after the steady state condition has been achieved (this time is specified for each fault in the faultload).
- 4) The detection time is dependent on the system features but it is also dependent on the type of faults. Furthermore, for some classes of faults such as operator faults, the detection time could be human dependent, as in some cases an operator faults can be only detected by the system administrator (i.e., another operator). In these cases it is necessary to assign a typical detection time, which has been estimated taking into account the nature of the fault and field experience in OLTP system administration.
- 5) After that detection time an error diagnostic procedure has to be executed to evaluate the effects of the fault and the required recovery procedure started (if an error is detected).
- 6) The recovery time represents the time needed to execute the recovery procedure. If no error is detected or no recovery is needed, then the recovery time is not considered (equal to zero).
- 7) When the recovery procedure completes, the workload must continue to run during a keep time in order to evaluate the system speedup after recovery.
- 8) After the workload end, a set of application consistency tests must be run to check possible data integrity violations caused by the fault injected. The integrity tests are performed on the application data (i.e., the data in the database tables after running the workload) and use both business rules and the database metadata to assure a comprehensive test.

It is worth noting that the duration of each injection slot depends on the fault injected and correspondent times (steady state time, injection time, detection time, recovery time, and keep time). However, the workload must run for at least 15 minutes after the steady state condition has been achieved, to assure the database run under realistic conditions concerning memory and disk accesses.

3.2 Workload

The DBench OLTP dependability benchmark adopts the workload of the well-established TPC-C performance benchmark, which represents a typical database installation. The business represented by TPC-C is a wholesale supplier having a number of warehouses and their associated sale districts, and where the users submit transactions that include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. This workload includes a mixture of read-only and update-intensive transactions that simulate the activities of most OLTP application environments, including transactions resulting from human operators working on interactive sessions. The TPC-C workload is submitted by the external driver system, which emulates all the terminals and their emulated users during the benchmark run.

3.3 Measures

The DBench OLTP dependability benchmark measures are computed from the information collected during the benchmark run and 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 comparison or for system/component improvement and tuning. 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 characterize system dependability in a relative fashion (e.g., to compare two alternative systems) 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.
- Allow the characterization of both dependability and performance features.
- Are easy to understandable (in both dependability and performance aspects) by database users and database administrators.

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 characterization of performance and dependability from the end-user point of view.

The DBench-OLTP measures are divided in three groups: baseline performance measures, performance measures in the presence of the faultload, and dependability measures.

The **baseline performance measures** are inherited from the TPC-C performance benchmark and are obtained during Phase 1. These measures include the number of transactions executed per minute (tpmC) and price per transaction (\$/tpmC). The number of transactions executed per minute represents the total number of completed New Order transactions (one of the 5 types of TPC-C transactions) divided by the elapsed time of the measurement interval. The price per transaction is a ratio between the price and the SUT performance (the system price is calculated based in a set of pricing rules provided in TPC-C specification and includes hardware, software, and system maintenance for a 3 years period). In the context of the DBench-OLTP, these measures represent a baseline performance instead of optimized pure performance (as it is the case of TPC-C), and should consider a good compromise between performance and dependability.

The **performance measures in the presence of the faultload** are:

- *Tf*: Number of transactions executed per minute in the presence of the faultload during Phase 2 (measures the impact of faults in the performance and favors systems with higher capability of tolerating faults, fast recovery time, etc).
- *\$/Tf*: Price-per-transaction in the presence of the faultload during Phase 2 (measures the relative benefit of including fault handling mechanisms in the target systems in terms of the price).
- *Tf/tpmC*: System performance decreasing ratio due to faults.

The **dependability measures** reported are:

- *Ne*: Number of data errors detected by the consistency tests and metadata tests (measures the impact of faults on the data integrity).
- *AvtS*: Availability from the SUT point-of-view in the presence of the faultload during Phase 2 (measures the amount of time the system is available from the SUT point-of-view). The system is available when it is able to respond at least to one terminal within the minimum response time defined for each type of transaction by the TPC-C benchmark. The system is unavailable when it is not able to respond to any terminal.
- *AvtC* – Availability from the end-users (terminals) point-of-view in the presence of the faultload during Phase 2 (measures the amount of time the system is available from the client's point-of-view). The system is available for one terminal if it responds to a submitted transaction within the minimum response time defined for that type of transaction by the TPC-C benchmark. The system is unavailable for that

terminal if there is no response within that time or if an error is returned.

It is worth noting that in the context of the DBench-OLTP dependability benchmark, availability is defined based on the service provided by the system. This way, the system is considered available when it is able to provide the service defined by the transactions. For example, from the client's point-of-view the system is not available if it submits a transaction and gets no answer within the specified time (see transaction profile in TPC-C specification [TPC-C 02]) or gets an error. In this case, the unavailability period is counted from the moment when a given client submits a transaction that fails until the moment when it submits a transaction that succeeds. From the server point of view, the system is available when it is able to execute transactions submitted by the clients. The measures *AvtS* and *AvtC* are given as a ratio between the amount of time the system is available and the Phase 2 duration.

3.4 Faultload

The faultload represents a set of faults and stressful conditions that emulates real faults experienced by OLTP systems in the field. A faultload can be based on three major types of faults: operator faults, software faults, and hardware faults. Although some of the published studies on the analysis of computer failures in the field are not directly focused on transactional systems, available studies clearly point operator faults and software faults as important causes for computer failures [Gray 90, Sullivan 91, Lee 95, Kalyanakrishnam 99, Sunbelt 99].

Operator faults in database systems are database administrator mistakes. The great complexity of database administration tasks and the need of tuning and administration in a daily basis, clearly explains why operator faults (i.e., wrong database administrator actions) are prevalent in database systems.

Concerning software faults, in spite of being considered an important source of failures, the emulation of software faults is still a research issue and there are no practical methods (at least well established enough to be used in a real dependability benchmark) readily available to inject this kind of faults [Christmansson 96, Madeira 00, Durães 02]. Thus we decided to include only operator faults in this first dependability benchmark proposal.

3.4.1 Operator faults in DBMS

The problem of operator faults in OLTP systems is essentially a problem of database administrator mistakes. End-user errors are not considered, as the end-user actions do not affect directly the dependability of DBMS. Database administrators manage all aspects of DBMS. In spite of constant efforts to introduce self-maintaining and self-administering features in DBMS, database administration still is a job heavily based on human operators.

The injection of operator faults in a DBMS can be easily achieved by reproducing common database administrator mistakes. That is, operator faults can be injected in the system by using exactly the same means used in the field by the real database administrator.

Different DBMS include different sets of administration tasks and consequently have different sets of possible operator faults. However, as shown in [Vieira 02b], it is possible to establish equivalence among many operator faults in different DBMS. In other words, a faultload based on operator faults is fairly portable across typical OLTP systems (see [Vieira 02b] for a detailed discussion on operator faults portability in three leading DBMS, respectively Oracle 8i, Sybase Adaptive Server 12.5, and Informix Dynamic Server 9.3). Furthermore, operator faults also emulate the high-level hardware failures (e.g., disk failures, network failures, etc) normally found in OLTP environments.

3.4.2 Faultload definition

The types of faults considered for the faultload have been chosen based on a estimation of the rate of occurrence, ability to emulate the effects of other types of faults (to improve the faultload representativeness), diversity of impact in the system, complexity of required recovery, and portability. The faultload is composed by a number of faults from these types, injected in different instants (i.e., with different injection times). Table 1 summarizes the faultload definition steps. Detailed guidelines to define

and implement the faultload are provided in the benchmark specification [Vieira 02c].

As we can see, the faultload depends mainly on the size and configuration of the data storage of the system under benchmarking (mainly, the files and disks configuration). This way, for systems with identical data storage configurations (in terms of the number and size of files and disks) the faultload to consider is exactly the same. For instance, the number of faults to inject from the type *Delete all files from one disk* depends only on the number of disks used, which means that two systems with the same number of disks will have the same number of faults of this type.

It is important to note that the comparison between systems of very different sizes is not a goal of the DBench-OLTP dependability benchmark. This way, in a given benchmarking campaign the faultload to use is normally identical.

3.5 DBench-OLTP specification overview

The DBench-OLTP benchmark consists of a specification defined in the form of an addendum to the TPC-C standard benchmark (see [Vieira 02c]). In order to run the DBench-OLTP dependability benchmark it is necessary to implement the TPC-C workload in the target system and the new benchmark elements (new measures and faultload) defined in the DBench-OLTP specification.

The DBench-OLTP specification follows the well accepted style of the TPC-C standard specification, and is

Type of fault	Target	Detection time
Abrupt operating system shutdown	Ten faults injected at the following injection times: 3, 5, 7, 9, 10, 11, 12, 13, 14, and 15 minutes.	0 Seconds
Abrupt transactional engine shutdown	Ten faults injected at the following injection times: 3, 5, 7, 9, 10, 11, 12, 13, 14, and 15 minutes.	30 Seconds
Kill set of user sessions	Five faults injected at the following injection times: 3, 7, 10, 13, and 15 minutes. The set of sessions to be killed in each fault injection must be randomly selected during the benchmark run and consists of 50% of all the active sessions from the users holding the TPC-C tables.	–
Delete table	Three faults for each one of the following TPC-C tables: ware, order, new-order, and order-line (a total of 12 faults). The injection times to be considered are the following: 3, 10, and 15 minutes.	2 Minutes
Delete user schema	Three faults using the following injection times: 3, 10, and 15 minutes. The user to be considered is the one that holds the TPC-C tables. If the objects are distributed among several users then the user holding the greater number of TPC-C tables must be selected.	1 Minute
Delete file from disk	The set of faults to inject is defined performing the following steps: For each TPC-C table: 1) Select randomly 10% of the disk files containing data from the TPC-C table being considered (in a minimum of 1). 2) Inject 3 faults for each disk file selected before, using the following injection times: 3, 10, and 15 minutes.	4 Minutes
Delete set of files from disk	Three faults for each set of files containing each TPC-C table (a total of 27 faults). The injection times are the following: 3, 10, and 15 minutes.	2 Minutes
Delete all files from one disk	The set of faults to inject is defined performing the following steps: 1) Select randomly 10% of the disks containing data from any TPC-C table (in a minimum of 1). 2) Inject 3 faults for each disk selected before, using the following injection times: 3, 10, and 15 minutes.	1 Minute

Table 1 – Faultload definition guidelines.

structured in clauses that define and specify how to implement the different components of the benchmark. Briefly, the structure of the DBench-OLTP dependability benchmark specification is as follows:

- *Clause 1. Preamble:* This clause provides an introduction to the DBench-OLTP benchmark and to the benchmark specification.
- *Clause 2. Benchmark Setup:* The benchmark setup is presented in this clause. The following elements of the setup are defined: Test configuration, System Under Test (SUT), Driver System, and Driver System/SUT Communications Interface.
- *Clause 3. Benchmark Procedure:* The benchmarking procedure, the phase 1 and phase 2 requirements, and the integrity testing requirements are presented in clause 3.
- *Clause 4. Measures:* This clause defines the measures provided by the DBench-OLTP benchmark and gives some guidelines on how to compute those measures.
- *Clause 5. Faultload:* Clause 5 presents the fault types that compose faultload and provides detailed guidelines to define and implement the faultload. The steps needed to inject the faults are also presented.
- *Clause 6. Full Disclosure Report:* Clause 6 specifies what needs to be included in the full disclosure report. Like in TPC-C performance benchmark, the DBench-OLTP benchmark requires that all the aspects concerning the benchmark implementation are disclosed together with the benchmark results.

To implement the DBench-OLTP dependability benchmark, existing code and examples can be adapted to new target systems, which greatly simplify the implementation process. This way, following the spirit of benchmarking, in which it is very important to reproduce the experiments (in other sites, in other systems, etc), the DBench-OLTP benchmark implementation used in this work is available at [<http://www.dei.uc.pt/~henrique/DBenchOLTP.htm>]. This implementation must be used together with the TPC-C implementation for Oracle, following the specification available at [TPC-C 02].

4. Dependability benchmarking examples using DBench-OLTP

The benchmarking examples presented in this section have resulted from an extensive study meant to validate the key properties of the proposed dependability benchmark. All the systems under benchmarking represent quite realistic alternatives for small and medium size OLTP applications.

Table 2 shows the systems under benchmarking (letters in the most left column will be used later to refer to each system). Two different versions of the Oracle DBMS (Oracle 8i and Oracle 9i), three different operating systems (Windows 2000, Windows Xp, and SuSE Linux 7.3), and two different hardware platforms (one based on a 800 MHz Pentium III with 256 MB of RAM and the other on a 2 GHz Pentium IV with 512 MB of RAM) have been used. Concerning the DBMS, two different configurations have been considered for each DBMS version. The main difference between these two configurations is that one provides better recovery capabilities (Configuration A) than the other (Configuration B). These configurations have been chosen based on results from a previous work on the evaluation of the Oracle recovery mechanisms in the presence of operator faults [Vieira 02a].

As mentioned earlier in the paper, the number of faults in the faultload is dependent on the size and configuration of the data storage of the system under benchmarking. In the present benchmarking experiments the configuration of the data storage is similar for all systems (the size of the database tables and the distribution of files among the available disks is almost the same). This way, the faultload used to benchmark a given system has exactly the same number of faults (and all the faults are equivalent) of the faultload used in the other. Table 3 summarizes that faultload.

The following sub-sections present and discuss the results of the benchmarking process conducted. The results are presented in a way that compares different alternatives for each one of the main components that compose a transactional system (the hardware platform, the operating

System	Operating System	DBMS	DBMS Config.	Hardware
A	Windows 2000 Prof. SP 3	Oracle 8i R2 (8.1.7)	Configuration A	<ul style="list-style-type: none"> • <i>Processor:</i> Intel Pentium III 800 MHz • <i>Memory:</i> 256MB • <i>Hard Disks:</i> Four 20GB / 7200 rpm • <i>Network:</i> Fast Ethernet
B	Windows 2000 Prof. SP 3	Oracle 9i R2 (9.0.2)	Configuration A	
C	Windows Xp Prof. SP 1	Oracle 8i R2 (8.1.7)	Configuration A	
D	Windows Xp Prof. SP 1	Oracle 9i R2 (9.0.2)	Configuration A	
E	Windows 2000 Prof. SP 3	Oracle 8i R2 (8.1.7)	Configuration B	
F	Windows 2000 Prof. SP 3	Oracle 9i R2 (9.0.2)	Configuration B	
G	SuSE Linux 7.3	Oracle 8i R2 (8.1.7)	Configuration A	
H	SuSE Linux 7.3	Oracle 9i R2 (9.0.2)	Configuration A	
I	Windows 2000 Prof. SP 3	Oracle 8i R2 (8.1.7)	Configuration A	<ul style="list-style-type: none"> • <i>Processor:</i> Intel Pentium IV 2 GHz • <i>Memory:</i> 512MB • <i>Hard Disks:</i> Four 20GB / 7200 rpm • <i>Network:</i> Fast Ethernet
J	Windows 2000 Prof. SP 3	Oracle 9i R2 (9.0.2)	Configuration A	

Table 2 – Systems under benchmarking.

system, and the DBMS) and for the DBMS configuration. The last sub-section presents a summary of the results and a comparison among the systems under benchmarking.

It is important to note that the system prices used to calculate the price per transaction presented are based in the set of pricing rules provided in TPC-C specification [TPC-C 02]. However, the prices considered in this benchmarking process are approximated prices and serve only as reference to compare the systems under benchmarking.

Type of fault	# of faults	% of faults
Abrupt operating system shutdown	10	10.3
Abrupt transactional engine shutdown	10	10.3
Kill set of user sessions	5	5.2
Delete table	12	12.4
Delete user schema	3	3.1
Delete file from disk	27	27.8
Delete set of files from disk	27	27.8
Delete all files from one disk	3	3.1
Total	97	100

Table 3 – Faultload used in the experiments.

4.1 Different operating systems and DBMS

Figure 3 shows the results regarding six different transactional systems using two versions of the Oracle DBMS (Oracle 8i and Oracle 9i), three different operating systems (Windows 2000, Windows Xp, and SuSE Linux 7.3), and the same hardware platform (systems A, B, C, D, G, and H from Table 2).

As we can see, results show that the baseline performance (tpmC) depends both on the DBMS and on the operating system used. In fact, a considerable difference in the baseline performance is observed for systems based in different types of operating systems (systems using Windows achieve a better number of transactions executed per minute than systems using SuSE

Linux). For the systems based on the SuSE Linux operating system, the baseline performance is similar independently of the DBMS used. On the other hand, for the systems based on Windows the baseline performance depends mainly on the DBMS and the same DBMS running over different Windows operating systems present a similar baseline performance. In terms of the price per transaction (\$/tpmC), and in spite of being less expensive, the systems based on SuSE Linux present the higher prices per transaction (due to the poor performance reached). Considering only systems running Windows, the more expensive ones (based on the Oracle 9i DBMS) present a lower price per transaction than the less expensive ones (based on the Oracle 8i DBMS), due to the better performance achieved.

Concerning the performance measures in the presence of faults, results show that the number of transactions executed per minute (Tf) also depends on the operating system and on the DBMS used. For the systems running the Oracle8i DBMS, Windows Xp is clearly more effective than Windows 2000 and for the systems running the Oracle 9i DBMS the reverse seems to occur (however, the small difference in the results for the systems using the Oracle 9i DBMS does not allow a solid conclusion). On the other hand, and as happened with baseline performance, the transactional systems based on the SuSE Linux operating system present similar results, independently of the DBMS used (which are also the worst results).

In terms of the price per transaction (\$/Tf), the less expensive systems (systems A, G and H) have the worst results (due to their poor performance in the presence of faults) and the two systems based on the Oracle 9i over Windows, although of having the same \$/tpmC, present quite different prices per transaction in the presence of faults.

A very interesting aspect to observe is that the ratio

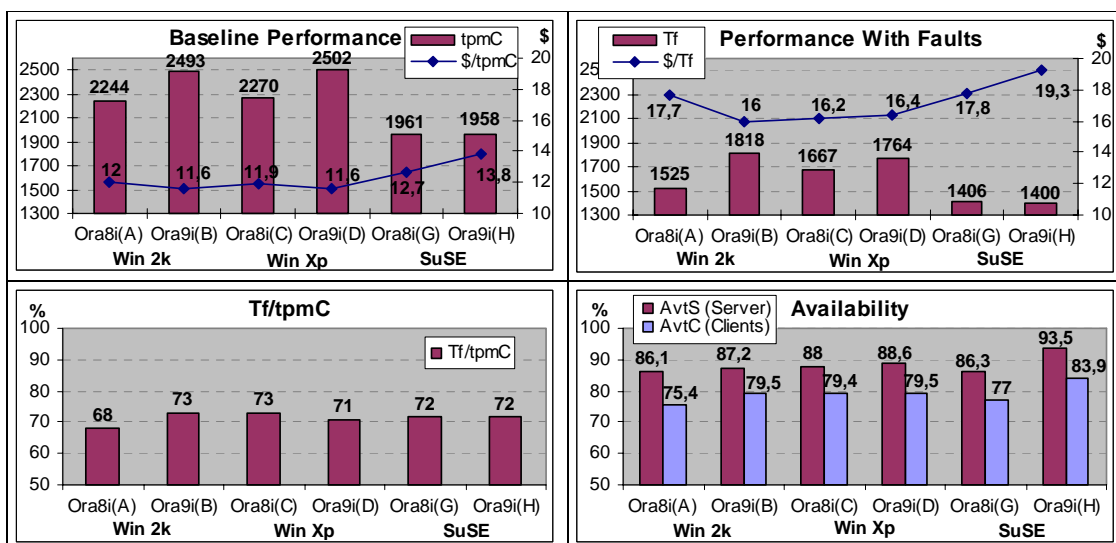


Figure 3 – Benchmarking results for systems using different DBMS and operating systems.

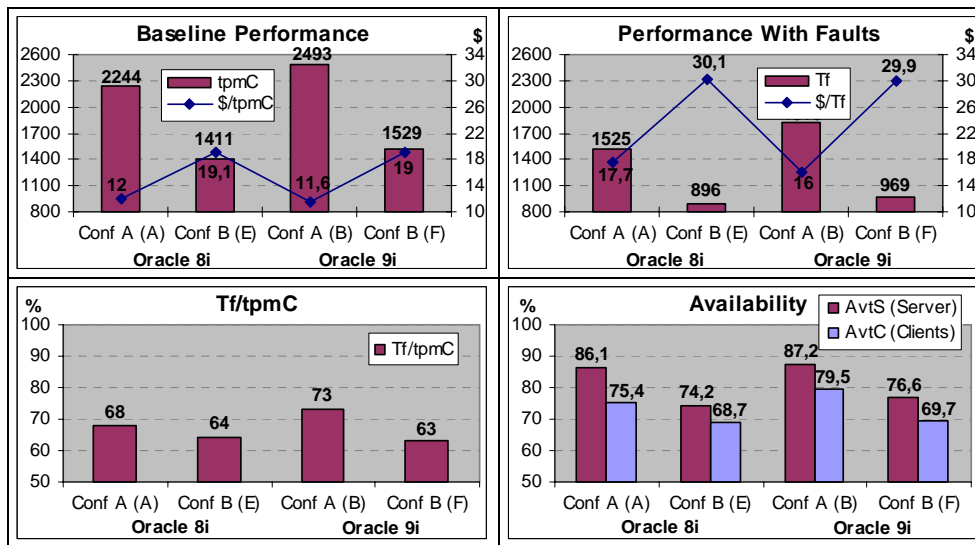


Figure 4 – Benchmarking results for systems using two different DBMS configurations.

between the Tf and tpmC (Tf/tpmC) is equal for the systems B and C, which shows that faults have a similar impact in a system with the Oracle 8i DBMS over Windows Xp and in a system with the Oracle 9i DBMS over Windows 2000. For the other four systems, the impact of faults on performance is more visible (the Tf/tpmC ratio is smaller).

Regarding the dependability measures, results show that the availability observed for the systems running the Oracle8i DBMS over Windows Xp is better than over SuSE Linux, which in turn is better than Windows 2000. Considering only Windows operating systems, a similar result has been observed for the systems running the Oracle 9i DBMS. For the system based on Oracle 9i running over SuSE Linux (system H), the availability is much higher than for any other system, which means that, although of being a slow system, it recovers from faults faster than the others (increasing the unavailability time).

An important aspect concerning the dependability features of the systems is that no data integrity errors (Ne) were detected, which shows that the Oracle transactional engine is very effective in handling faults caused by the operator.

4.2 Different DBMS configurations

Figure 4 compares four different transactional systems using two versions of the Oracle DBMS (Oracle 8i and Oracle 9i) running over the Windows 2000 operating system and using the same hardware platform (systems A, B, E, and F from Table 2). In these experiments, each one of the Oracle DBMS was tested using two different configurations of the recovery mechanisms. The main difference between these two configurations is that one provides better recovery capabilities (Configuration A) than the other (Configuration B). As mentioned before, these configurations have been chosen based on results from a previous work [Vieira 02a]. Results show that

Configuration A is better than Configuration B in both the Oracle 8i and Oracle 9i DBMS.

It is worth noting, that the use of Configuration B in the Oracle 8i DBMS leads to smaller losses (comparatively to Configuration A) than in the Oracle 9i DBMS. For instance, while the tpmC decreases 37.1% from Configuration A to Configuration B in Oracle 8i, it decreases 38.7% in Oracle 9i. A similar behavior can be observed for all the other measures except AvtS. Table 4 summarizes the results for Configuration A and Configuration B in both DBMS and shows the results variation from configuration A to Configuration B (note that the increase of \$/tpmC and \$/Tf represents an increasing of the price per transaction, which is a bad result).

Measures	Oracle 8i			Oracle 9i		
	Conf. A	Conf. B	Var.	Conf. A	Conf. B	Var.
tpmC	2244	1411	-37,1 %	2493	1529	-38,7 %
\$/tpmC	12	19,1	+59,2 %	11,6	19	+63,8 %
Tf	1525	896	-41,2 %	1818	969	-46,7 %
\$/Tf	17,7	30,1	+70,1 %	16	29,9	+86,9 %
Tf/tpmC	68	64	-5,9 %	73	63	-13,7 %
AvtS	86,1	74,2	-13,8 %	87,2	76,6	-12,2 %
AvtC	75,4	68,7	-8,9 %	79,5	69,7	-12,3 %

Table 4 – Results variation using two different DBMS configurations.

4.3 Different hardware platforms

In order to assess the impact of the hardware platform in a transactional system, Figure 5 compares four different systems using two versions of the Oracle DBMS (Oracle 8i and Oracle 9i) running over the Windows 2000 operating system and using two different hardware platforms (systems A, B, I, and J from Table 2). The main differences between these two platforms are the CPU used and the amount of RAM available (one of the hardware platforms is based on a 800 MHz Pentium III with 256

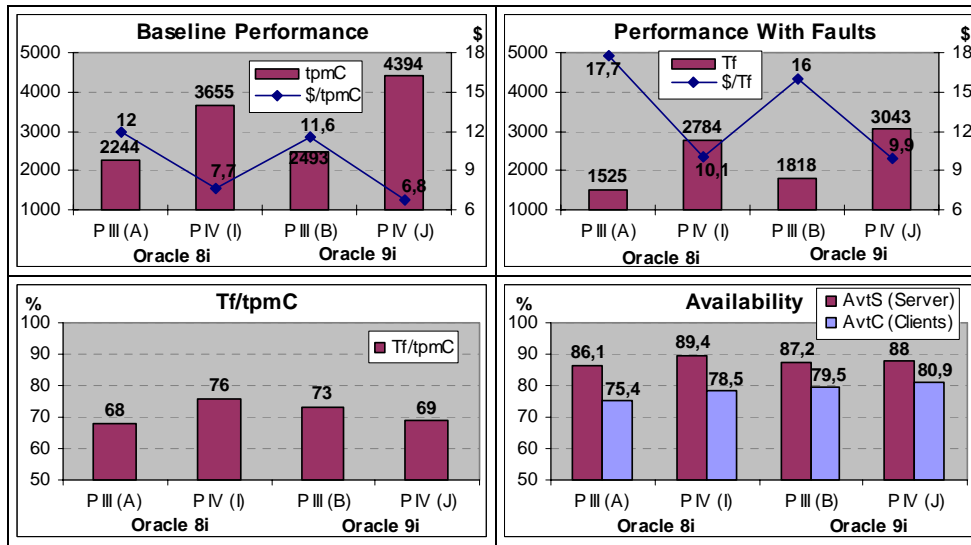


Figure 5 – Benchmarking results for systems using two different hardware platforms.

MB of RAM and the other is based on a 2 GHz Pentium IV with 512 MB of RAM). It is important to note that the DBMS has been configured to use different amounts of memory according to the size of the RAM available. As expected, results show that the hardware platform based on the Pentium IV presents better performance results (baseline and in the presence of faults) than the hardware platform based on the Pentium III. However, concerning dependability measures the hardware platform has some impact but it is not as visible as for the other measures.

4.4 Results summary

In the previous sub-sections, we have compared different alternatives for each one of the main components of an OLTP system (the hardware platform, the operating system, and the DBMS). In this sub-section we present a summary of the results and propose a ranking for the

systems under benchmarking.

Figure 6 shows the DBench-OLTP results for all systems (see Table 2 for the correspondence between the labels in the X axis and the systems under benchmarking). As we can see, the baseline performance and the performance in the presence of faults are strongly dependent on the hardware platform and DBMS configuration used. The DBMS and operating system have a lower impact.

An interesting result is that availability depends mainly on the DBMS configuration. In fact, systems with the same DBMS configuration present a similar level of availability, independently of the hardware platform, operating system and DBMS used. Another interesting result is that the availability from the clients point-of-view (AvtC) is always much lower than the availability from the server point-of-view (AvtS), which seems to be

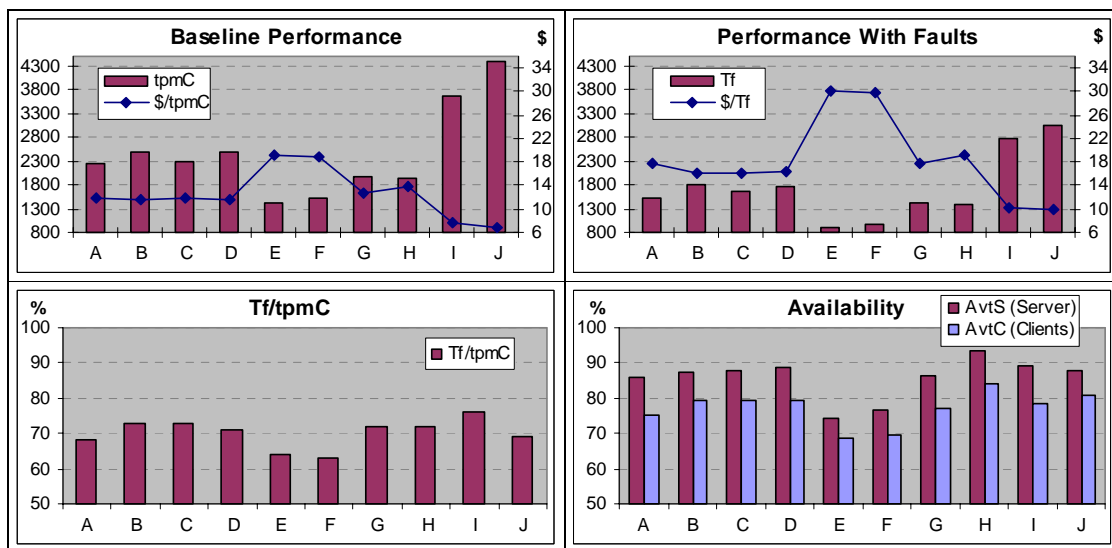


Figure 6 – Benchmarking results summary.

normal because some types of faults affect the system in a partial way (e.g., when a given file is removed from disk only the transactions that need to access to the data stored in that file are affected).

Table 6 summarizes the ranking proposed according to several criteria.

Criteria	System Ranking (best to worst)
Baseline performance (tpmC)	J, I, D, B, C, A, G, H, F, E
Performance with faults (Tf)	J, I, B, D, C, A, G, H, F, E
Ratio Tf/tpmC	I, B, C, G, H, D, J, A, E, F
Availability (AvtS and AvtC)	H, I, D, J, C, B, G, A, F, E

Table 5 – Systems ranking according to several criteria.

Concerning a global ranking, the analysis of Table 6 and all the results presented before allow us to propose the following order (from the best to the worst): I, J, D, B, C, H, G, A, F, and E. It is important to note that the global ranking always depends on the benchmark performer point-of-view (i.e., depends on what he is looking for).

5. Benchmark execution effort

Usually benchmarking is seen as an expensive and laborious process. During the course of the present work, we had the opportunity to assess the necessary effort to implement the benchmark and to conduct the benchmarking process. Several indicators have been collected, such as: the time needed to implement the TPC-C benchmark, the time needed to implement the DBench-OLTP benchmark, and the time needed to conduct the benchmarking process. Table 6 summarizes the observations in terms of the number of working days of one experienced person.

Type of fault	# of days
TPC-C benchmark implementation	10
DBench-OLTP benchmark implementation	10
Benchmarking process execution	30
Total time	50
Average time per system	5

Table 6 – Benchmark execution effort.

As we can see, although being the most complex task, the implementation of the TPC-C benchmark takes only about 10 days. This was possible due to the reuse of existing code and examples from several previous implementations. In a normal situation the TPC-C implementation alone could take more than 30 working days.

Comparing to TPC-C, the DBench-OLTP benchmark presents a similar implementation time. However, as for TPC-C we can reduce the effort needed to implement this dependability benchmark by reusing code from previous implementations (in our case this was not possible because this was the first implementation of this benchmark).

Concerning the time needed to conduct the benchmarking process, the effort was very low, mainly because the benchmark run is fully automatic. In fact, considering the class of systems used in this work we have been able to benchmark ten different systems in about one month. The ratio between the total effort and number of systems benchmarked is of about 5 working days. However, it is important to note that this ratio decreases when the number of systems under benchmarking increases (e.g., if instead of having benchmarked ten transactional systems we had benchmarked twenty, the average time would decrease from 5 to about 4 working days). Thus, we can conclude that after having the benchmark implemented (TPC-C and DBench-OLTP) the effort needed to benchmark additional systems is relatively small.

6. Conclusions

This paper proposes a new dependability benchmark for OLTP application environments – the DBench-OLTP dependability benchmark. This benchmark specifies the measures and all the steps required to evaluate both the performance and key dependability features of OLTP systems. The DBench-OLTP uses the basic setup, the workload, and the performance measures specified in the TPC-C performance benchmark, and adds two new elements: 1) measures related to dependability; and 2) a faultload based on operator faults.

Several different transactional systems have been benchmarked using the DBench-OLTP benchmark. Two different versions of the Oracle DBMS (Oracle 8i and Oracle 9i), three different operating systems (Windows 2000, Windows Xp, and SuSE Linux 7.3), and two different hardware platforms (one based on a 800 MHz Pentium III with 256 MB of RAM and the other on a 2 GHz Pentium IV with 512 MB of RAM) have been used. Concerning the DBMS, two different configurations have been considered for each DBMS version. The results obtained were analyzed and discussed in detail. These results allowed us to rank the systems under benchmarking concerning both performance and dependability and clearly show that dependability benchmarking can be successfully applied to OLTP application environments.

The paper ends with a discussion on the effort required to run the DBench-OLTP dependability benchmark. From the indicators collected during this work, we could observe that that effort is not an obstacle for not using this kind of tools on small and medium size transactional systems evaluation and comparison.

7. References

[Christmansson 96] J. Christmansson and R. Chillarege, "Generation of an Error Set that Emulates Software

Faults”, Proceedings of the 26th IEEE Fault Tolerant Computing Symposium, FTCS-26, Sendai, Japan, pp. 304-313, June 1996.

[Gray 90] J. Gray, “A Census of Tandem Systems Availability Between 1985 and 1990”, IEEE Transactions on Reliability, Vol. 39, No. 4, pp. 409-418, October 1990.

[Gray 93] J. Gray (Ed.), “The Benchmark Handbook”, Morgan Kaufmann Publishers, San Francisco, CA, USA, 1993.

[Hsueh 97] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer, Fault Injection Techniques and Tools, IEEE Computer, 30(4), pp. 75-82, 1997.

[Jenn 95] E. Jenn, J. Arlat, M. Rimén, J. Ohlsson and J. Karlsson, “Fault Injection into VHDL Models: The MEFISTO Tool”, in Predictably Dependable Computing Systems (B. Randell, J.-C. Laprie, H. Kopetz and B. Littlewood, Eds.), pp.329-46, Springer, Berlin, Germany, 1995.

[Kalyanakrishnam 99] M. Kalyanakrishnam, Z. Kalbarczyk, R. Iyer, “Failure Data Analysis of a LAN of Windows NT Based Computers”, Symposium on Reliable Distributed Database Systems, SRDS18, October, Switzerland, pp. 178-187, 1999.

[Koopman 99] 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.

[Lee 95] I. Lee and R. K. Iyer, “Software Dependability in the Tandem GUARDIAN System”, IEEE Transactions on Software Engineering, Vol. 21, No. 5, pp. 455-467, May 1995.

[Madeira 00] H. Madeira, M. Vieira and D. Costa, “On the Emulation of Software Faults by Software Fault Injection,” Intl. Conf. on Dependable Systems and Networks, New York, USA, June, 2000, pp. 417-426.

[Madeira 01] H. Madeira, K. Kanoun, J. Arlat, Y. Crouzet, A. Johanson and R. Lindström, “Preliminary Dependability Benchmark Framework”, DBench Project, IST 2000-25425, August 2001.

[Madeira 02] H. Madeira et al., “Dependability Benchmark Definition: DBench Prototypes,” DBench Project IST 2000-25425 Deliverable no. BDEV1, Available at <http://www.laas.fr/dbench/deliverables.html>, 2002.

[Ramakrishnan 00] R. Ramakrishnan, “Database Management Systems” second edition, McGraw Hill, ISBN 0-07-232206-3, 2000.

[Siewiorek 93] D. P. Siewiorek, J. J. Hudak, B.-H. Suh and Z. Segall, “Development of a Benchmark to Measure System Robustness”, in Proc. 23rd Int. Symp. on Fault-Tolerant Computing (FTCS-23), (Toulouse, France), pp.88-97, IEEE CS Press, 1993.

[Sullivan 1991] M. Sullivan and R. Chillarege, “Software defects and their impact on systems availability – A study of field failures on operating systems”, Proceedings of the 21st IEEE Fault Tolerant Computing Symposium, FTCS-21, pp. 2-9, June 1991.

[Sunbelt 99] Sunbelt Int., “NT Reliability Survey Results”, <http://www.sunbelt-software.com/ntrelres3.htm>, published on March, 23, 1999.

[TPC-C 02] Transaction Processing Performance Consortium, “TPC Benchmark C, Standard Specification, Version 5.1”, 2002, available at: <http://www.tpc.org/tpcc/>.

[Trivedi 94] K. S. Trivedi, B. R. Haverkort, A. Rindos and V. Mainkar, “Methods and Tools for Reliability and Performability: Problems and Perspectives”, in Proc. 7th Int'l Conf. on Techniques and Tools for Computer Performance Evaluation (G. Haring and G. Kotsis, Eds.), Lecture Notes in Computer Science, 794, pp.1-24, Springer-Verlag, Vienna, Austria, 1994.

[Vieira 02a] Marco Vieira and Henrique Madeira, “Recovery and Performance Balance of a COTS DBMS in the Presence of Operator Faults”, International Performance and Dependability Symposium (jointly organized with DSN-2002), IPDS2002, Bethesda, Maryland, USA, June 23-26, 2002.

[Vieira 02b] Marco Vieira and Henrique Madeira, “Definition of Faultloads Based on Operator Faults for DMBS Recovery Benchmarking”, 2002 Pacific Rim International Symposium on Dependable Computing, PRDC2002, Tsukuba, Japan, December 16-18, 2002.

[Vieira 02c] Marco Vieira and Henrique Madeira, “DBench – OLTP: A Dependability Benchmark for OLTP Application Environments”, Technical Report DEI-006-2002, ISSN 0873-9293, Departamento de Engenharia Informática – Faculdade de Ciências e Tecnologia da Universidade de Coimbra, 2002, available at: <http://www.dei.uc.pt/~henrique/DBenchOLTP.htm>.