

# Experimental Assessment of COTS DBMS Robustness under Transient Faults

Diamantino Costa<sup>†‡</sup>

CISUC—Centro de Informática e Sistemas da UC  
University of Coimbra, P-3030 Portugal  
dino@dei.uc.pt

Henrique Madeira

Departamento de Engenharia Informática  
University of Coimbra, P-3030 Portugal  
henrique@dei.uc.pt

## Abstract

*This paper evaluates the behavior of a common off-the-shelf (COTS) database management system (DBMS) in presence of transient faults. Database applications have traditionally been a field with fault-tolerance needs, concerning both data integrity and availability. While most of the commercially available DBMS provide support for data recovery and fault-tolerance, very limited knowledge was available regarding the impact of transient faults in a COTS database system. In this experimental study, a strict off-the-shelf target system is used (Oracle 7.3 server running on top of Wintel platform), combined with a TPC-A based workload and a software implemented fault injection tool – XceptionNT. It was found out that a non-negligible amount of induced faults - 13% - lead to database server hanging or premature termination. However, the results also show that COTS DBMS products has a reasonable behavior concerning data integrity - none of the injected faults affected end user data.*

## 1. Introduction

Database applications have traditionally been a field with fault-tolerance needs, concerning both data integrity and availability. Furthermore, several mechanisms needed to achieve fault-tolerance, such as transactions, checkpointing, log files, and replica control management, have been developed or improved on the databases behalf. The benefits that would come out of experimental fault-injection enabling validation, evaluation and fine tuning of those techniques are recognized by the database research community [1].

Most of the commercially available database management systems (DBMS) provide support for data recovery and fault-tolerance, even when the underlying hardware platforms do not have any fault-tolerance features. However, few works in the literature address the evaluation/validation of those fault-tolerant techniques

[2][3][4] and, to the best of our knowledge, no one has addressed yet the evaluation of the impact of transient faults in a common off-the-shelf (COTS) database system.

Most of the dependability enforcing solutions available in commodity DBMS address permanent faults and assume a fail-stop model for the system. Little or few interest has been given to transient hardware faults. While it is true that software and operations are becoming major sources of service disrupt, transient hardware faults still account for a non-negligible stake of computer system failures. These figures tend to have much more relevance in presence of COTS systems with no hardening features or specific build-in fault-tolerance mechanisms. The increasing trend of using COTS technology (DBMS included) on mission critical and business critical systems pushes even further the interest for dependability evaluation of such systems. Will COTS savings mean less robustness? And how much “less” it will be?

This paper gives an insight on the behavior in presence of transient faults of a DBMS that was strictly taken off-the-shelf. The target system consists of an Oracle 7.3 server running on top of WindowsNT 4.0 operating system on an Intel based P6 hardware platform, loaded over the network with “clients” running a TPC-A like benchmark [5].

One clear difficulty in the evaluation of the impact of faults in a COTS database is the huge complexity of existing DBMS. Software-implemented fault injection techniques (SWIFI) are probably the best alternative to accomplish this evaluation. However, the injection of transient faults in a complex DBMS, especially in a COTS system in which the source code is not available, is a difficult task. The goal of injecting low level faults that directly emulate transient hardware faults (and indirectly induce erroneous system behavior similar to the one induced by software errors) requires minimal intrusion of the SWIFI tool. The fault injection tool should not disturb the transaction timing, as the TPC benchmark used in the experiments emulates a typical interactive database application (which excludes running the server in trace

<sup>†</sup> Research supported in part by *Fundação para a Ciência e Tecnologia - PRAXIS XXI* under grant number BD/5636/95.

<sup>‡</sup> On leave from Critical Software, [www.criticalsoftware.com](http://www.criticalsoftware.com).

mode). On the other hand, the additional code required to implement the fault injection mechanism in the server should be as small as possible, to avoid the well known problem of the interference of the fault injection code in the fault impact. The fault-injection tool used is the XceptionNT SWIFI tool, which is a preliminary version of Xception [6] for the Windows NT operating system. XceptionNT takes advantage of Pentium processor and OS debugging resources to perform fault injection with minimal intrusion.

The impact of faults is assessed both at the operating system level and at the DBMS level. Particularly, we are especially interested in the impact of faults in the database integrity and in time-to-repair measurements. Database integrity is addressed at database application level, using a set of semantic rules and associated consistency tests (“business logic”).

The paper is organized as follow: the next section describes the related work. Section 3-Testbed details the testbed, including the target system, the workload, the set of faults and the set of measurements. Section 4-Experiment setup and execution addresses the dynamic traits of the testbed, describing experiment setup and the main steps of an injection run. Section 5-Results presents and discusses the experimental results. Section 6-Conclusions and Future work ends the paper, summing up the main contributions and presenting some research directions for the future.

## 2. Related work

A comprehensive survey on field outages and software errors (that originated them) in two commercial DBMS products with large distribution (IMS and DB2) is made in [2]. Several techniques, derived from error analysis, were then proposed for early error detection and fast recovery in the POSTGRES DBMS [7].

A substantial research work cross relating fault-injection and databases has been made in the RIO project. Part of this work was targeted to show that non-volatile memory (NVRAM) could be as reliable as disk storage in presence of application and OS failures. In fact, with prices of NVRAM memory going down (and densities up) and disk throughput rates increasing smoothly, one could expect that if primary memory subsystems prove to rate not far from disks em terms of reliability figures, there will be candidates for trading off robustness by speed. In [3], the evaluation of memory resistance to operating system software crashes induced by faults estimated that only around 2% of the faults lead to data corruption in the file cache.

In [4], a simulation study was made to measure the contribution of transactions to the fail-stop behavior of a POSTGRES installation. In that work, which shares almost the same infrastructure as the above-mentioned RIO experimental study, was conclude that 7% of the faults

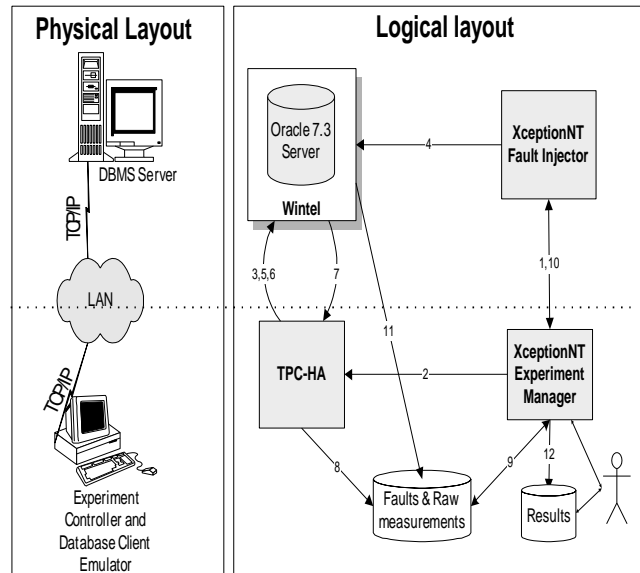
caused fail-stop violations (in the sense that faulty data was written to stable storage) without transactional support. When transactions were added, this figure drops to 2%.

## 3. Testbed

Figure 1 depicts the testbed layout used in this experimental evaluation. The key testbed components such as target system, workload, set of faults and the set of measurements are described in the following subsections. The fault-injection tool is XceptionNT, a port of Xception [6] to the WindowsNT/Pentium platform. Since Xception has already been used extensively in the past, no dedicated session is used to describe the intricacies of the Xception fault-injection process. The role of XceptionNT in the experiments is described later on in section 4-*Experiment setup and execution*. Most of the dynamics aspects regarding the testbed will be also postponed to section 4. The following presentation will focus on static configuration.

### 3.1 Target system

Oracle™ have been chosen as target DBMS because it is widely used and it represents quite well a typical (and sophisticated) DBMS. The same reasons apply to the choice of a Wintel based platform. Table 1 details the target system configuration, together with data on OS and hardware infrastructure.



- |                               |                            |
|-------------------------------|----------------------------|
| 1. Fault parameters           | 7. “Probes” & measures     |
| 2. Start workload             | 8. Raw measurement data    |
| 3. Database gold files        | 9. Raw data and faults     |
| 4. Fault Injection            | 10. Fault impact readouts  |
| 5. Database instance shutdown | 11. Log and trace activity |
| 6. Database instance startup  | 12. Processed results      |

Figure 1. Testbed layout

Table 1. Target System Configuration

<b>Server</b>	Intel Pentium II processor based machine with 128 MB RAM
	Windows NT 4.0 Server build 1381 with SP3 Oracle 7 Server Release 7.3.3.0.0 PL/SQL Release 2.3.3.0.0 SQL*Net Server 2.3.3.0.0
<b>Client</b>	Intel Pentium processor based machine with 64 MB RAM WindowsNT 4.0 Workstation with SP3 SQL*Net Client 2.3.3.0.0
<b>Network</b>	10 Mbit Ethernet with TCP/IP

It is worthwhile to have a look at the internal architecture of Oracle 7.3 database engine, as it helps to understand the fault-injection process and the experiment results. One main technical aspect distinguishes the architecture of Oracle on WindowsNT from the UNIX relatives. Instead of a “bunch” of processes with shared memory making up the System Global Area (SGA), NT implementation lies on a unique OS process where multiple threads (or light-weight processes) share the SGA on a single address space.

The recovery is based on a redo log mechanism. The Log Writer process (LGWR) writes the redo log buffer to a redo log file on disk (see figure 2). Whenever a user process commits the LGWR process saves the redo entries that are in the redo buffer in the online redo log files (other events can also cause the redo entries to be written in disk). The online redo log files (in a minimum of two) work in a circular way and the archive process (ARCH) copies online redo log files to a tertiary storage device once they become full (only if this option is on). Oracle performs periodical checkpoints (several events can trigger checkpoints) which represents consistent states of the database from which it is possible to start recovery. The recovery consists in a forward phase, in which the redo entries recorded in the redo log are applied to regenerate the data (and other structures such as rollback segments), followed by a backward phase, in which unrecoverable transactions are rolled back to bring the database to a clean state.

### 3.2 Workload

Online transaction processing (OLTP) are definitely among the top most stringent workloads for DBMS in terms of dependability requirements. Typical corporate databases have a mix of batch and OLTP loads, but the later is responsible for most of the tuning effort for delivering proper service response for assuring ACID

properties as well as availability and performance. A typical example of such a scenario is a bank transaction, which inspired the TPC-A benchmark [5]. The workload used in this experiment is based on the TPC-A benchmark, which mimics a bank with multiple branches, each branch with accounts and tellers and with debit-credit like transactions. While probably TPC-C might be a much more convenient solution in terms of being even more close to a real scenario, the TPC-A was chosen for simplicity reasons (it is worth noting that all the setup is very complex, which justify the choice of a simple benchmark for the initial experiments). The transaction profile for TPC-A is depicted in Figure 3. The transaction procedure was deployed at the DBMS server using PL/SQL, making up the server part of the client/server workload.

The TPC-A benchmark was adapted (slightly) in order to match the fault-injection based evaluation environment. The resulting workload, coined TPC-HA [9], brings a new modeled entity – the database administrator (DBA) – and expands the terminal emulator program (TEP) with additional facilities to accommodate the fault injection process and its outcomes.

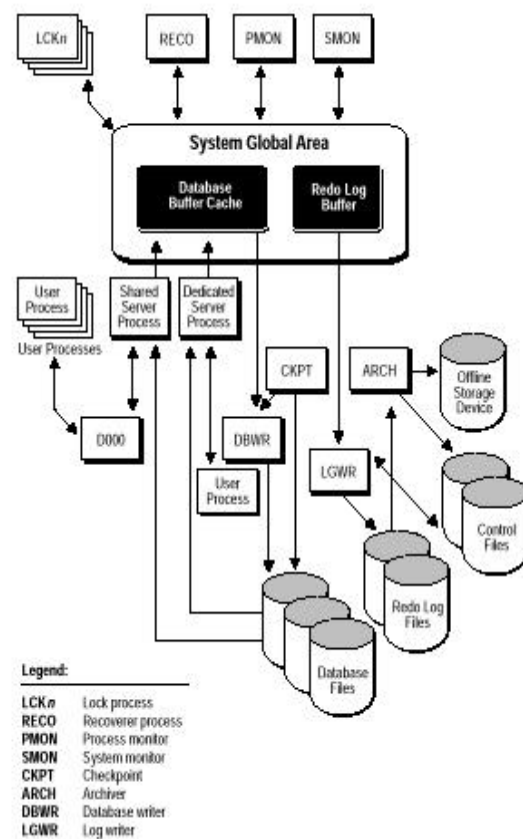


Figure 2. The Background Processes and recovery structures in an Oracle Instance [8]

The DBA entity, as well as the new features in the TEP, account for the presence of failures and pursue together the detection, diagnosis and recovery of failures in the client/server installation. Figure 4 illustrates the execution outline for the TPC-HA client part showing the concurrency between the end user and DBA emulators.

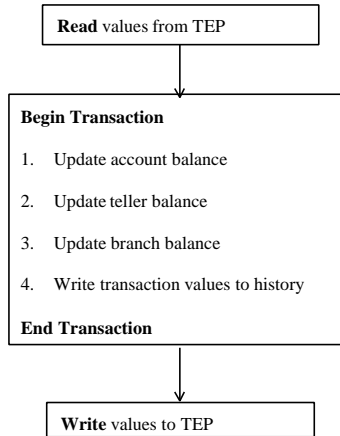


Figure 3. TPC-A transaction profile (server part of workload)

The TEP starts execution by populating the database schema (account, branches and tellers tables). The client application starts then its core activity, invoking repeatedly the server procedure until all the input transactions are submitted (1000 for this particular experiment) or a timeout condition is reached. If by any chance an error occurs during transaction submission or the invoking procedure itself times out, the current database session is closed (if it is not already) and a new connection is attempted with the DBMS server. On success, the TEP activity restarts from the point of failure. The consistency tests run when the TEP ceases its activity.

As it can be seen on Figure 4, the DBA part of the TPC-HA client workload will remain idle during the injection run, unless any trouble happens in TEP execution. The DBA thread rests waiting on transaction timeout events, and when one such happens it tries first to unblock the TEP session by closing it. If this is not enough to restart normal database activity, the DBA performs successive attempts to put the DBMS online, with increasing impact on the target system, ending with database process restart. The DBA does not directly initiates any database recovery procedure in the strict meaning of the term, i.e., bringing data to a consistent state. Database recovery, if needed, will be triggered by Oracle automatically when an instance starts up.

### 3.3 Faults

The set of faults applied in this experimental study is primarily meant to be representative of transient hardware faults affecting computer systems. However, it is generally

accepted that errors injected this way also emulate other kinds of faults, such as some classes of software faults (bugs). The analytical study in [10] clearly points in this direction. Still, it must be said that our study does not gather (nor it was meant to have) all the necessary conditions to address a software faults based evaluation; the lack of field data for this particular target system is perhaps the most significant shortcoming towards that purpose, among others. Nevertheless, one can expect that the results obtained, or at least the "big picture", will follow close the ones that would have been obtained from a set of faults with both hardware and software fault models. Indeed, the set of faults within this experiment are pretty much close to the two first categories of faults mentioned in [4], namely *hardware bit flips* and *low-level software faults*. From this later study, it can be seen that no significant differences were found on the impact of faults between the so-called high-level software faults and low-level faults.

The following paragraphs describe the set of faults in terms of location, type, trigger and duration and, at last, a summary table concludes this section.

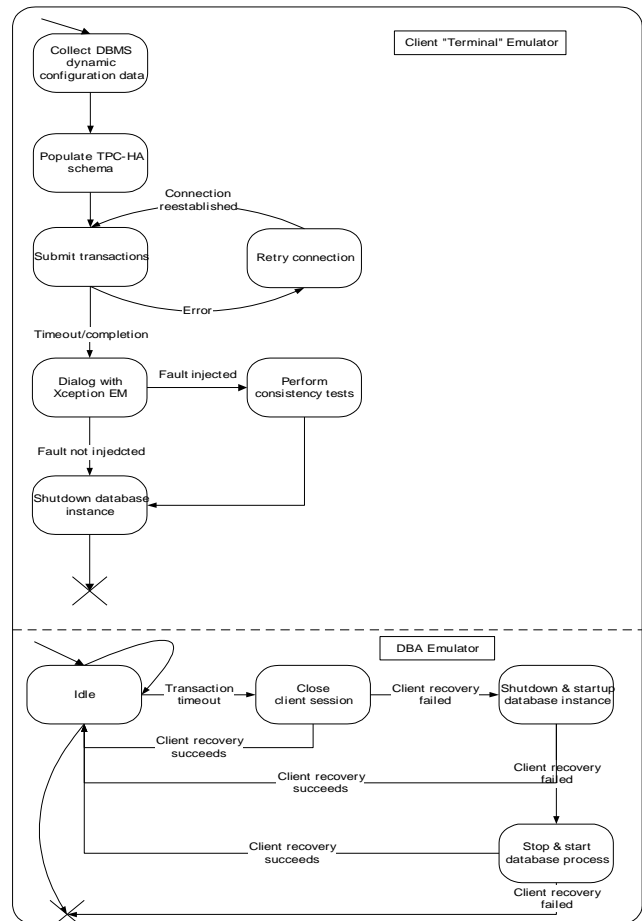


Figure 4. TPC-HA client execution outline

### 3.3.1 Location

In terms of broad location, faults are injected in the database processes/threads only. No attempt is here made to mimic client or network faults. Despite of being fairly easy to inject faults in middleware and client components we are more concerned with server faults since they represent the biggest impairment to system dependability [11]. In fact, while network failures might be as critical as server failures in terms of service availability to clients, server failures represent the most significant peril against corporate data integrity.

Narrowing the location perspective, faults were only injected in the Oracle process when running at user level mode. WindowsNT OS makes use of two x86 Intel architecture protection rings – ring 3 is used for user level code while kernel and device drivers code run at ring 0. This way we account for DBMS resilience itself, at the same time monitoring how OS handles system calls from faulty processes. While it is definitely interesting to perform the same evaluation targeted at faults affecting directly the kernel and other system components, and we are planning to do it in the near future, it was decided to focus on the DBMS product for this study.

The set of faults injected followed previous fault injection experiments made with the Xception tool, but this time we have affected only the CPU registers. At first sight, the register only model may seem to be too much restrictive. However, it should be stressed that the Pentium processor is far way from RISC "load store" memory access model. In fact, the X86 architecture is rich on instructions with mixed memory and register operands (and compilers use it a lot). This way, it is possible to interfere with memory operands and memory access (address generation included) by manipulating register contents.

### 3.3.2 Type

Only bit-flip faults were considered in order to accelerate the production of effective injection runs, and only one bit, choose at random, is affected per fault. It is our belief that, besides being fairly matched with real hardware transient faults, this model also accounts for the most subtle errors and hard to detect (including some software errors).

### 3.3.3 Trigger

Faults were triggered based on time. A randomly chosen time interval set between DBMS process start time and expected workload duration was used to fire up fault-injection. This way, faults were distributed along execution time affecting potentially execution of Oracle static code or any loadable module (DLL) code present.

### 3.3.4 Duration

As was already pointed out this paper deals exclusively with transient faults, i.e., faults with very small duration,

typically within the same order of magnitude of CPU processor clock cycle period. In the case of this particular experiment, since the set of faults is targeted at CPU registers and CPU internal busses fault duration is somehow equivalent to the time spent on the execution of the *target* instruction. The target instruction is the one whose machine execution context is corrupted by the fault, i.e., on which context the injector code will modify the target register according with the fault mask. This way, if after the target instruction has been executed, a new value is written into the target register the (old) corrupted value will be rewritten.

Table 2. Set of faults

Location	Type	Trigger	Duration
CPU registers; {EAX,...,EDX, BP; SP, Eflags, IP, CS, DS, ES, FS, GS, SI, DI}	Bit-flip. One bit at a time.	[1..60] * 10 <sup>6</sup> μs. Start of time is DBMS process first instruction	1 CPU instruction

## 3.4 Readouts and error and failure detection techniques

The error and failure detection mechanisms evaluated on this experimental study, as well as monitored events, can be classified along two main criteria: level of abstraction and if they are built-in or not in the target system. Table 3 enumerates them according to these criteria. They all run for each injection run where a fault has been injected.

Possible machine level exceptions raised within the Oracle process as well as exit code values are saved for each injection run. The later determines if the Oracle had terminated normally or abnormally. By default, on the Win32 platform certain kinds of exceptions (memory access violations, illegal instructions) force the termination of the issuing thread, which in turn might cause the owner process to terminate.

Table 3. Error and failure detection and monitoring techniques

Name	Abstraction level	Built-In (Yes/No)
Exception handling	OS	Yes
Exit code	OS	Yes
TPC-HA consistency tests	DBMS/Application	No
User & DBA "watchdog"	DBMS/User	No

TPC-HA consistency tests are defined according with the clauses on the standard [5] and, basically, they perform an evaluation of the consistency of the "user" data hold in the database – these kind of rules are often referenced as "the business logic". For TPC-A the following rules apply:

1. The sum of the account balances is equal to the sum of the teller balances, which is equal to the sum of the branch balances.

2. For all branches, the sum of the teller balances within a branch is equal to the branch balance.
3. The sum of the “deltas” in the history file is equal to the sum of the branch balances.

As was already mentioned, the *user & DBA watchdog* plays the role of an application end user, triggering a "local" recovery procedure when a transaction freezes. Both the triggering event and the first transaction resubmitted with success are time-stamped together with other system behavior traits to a log file. This log is

The sequence of events making one injection run is illustrated on Figure 5. Several steps and variations were not included on purpose in the figure in order to make it simple to understand the essential. In order to avoid contamination between the effects of successive faults, the target system is reset before starting each fault injection run. While it seems questionable that faults injected at processor “user” level could actually corrupt kernel status, it was decided to perform a complete system reset, so that no contamination was possible between subsequent

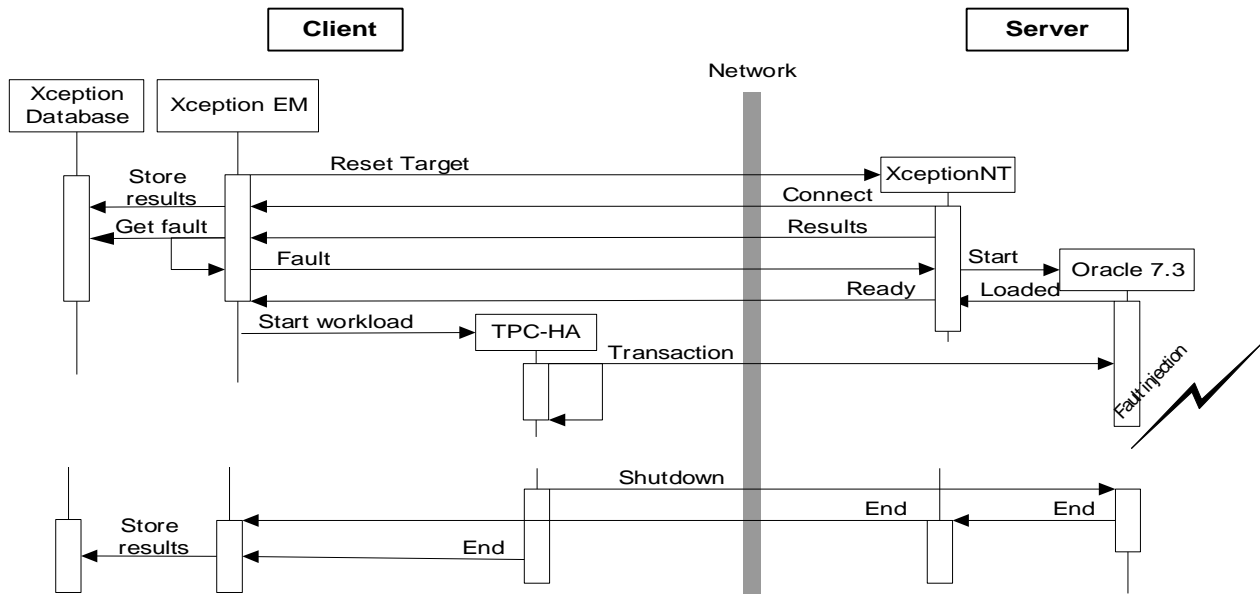


Figure 5. Injection run sequence diagram

extremely useful to understand the behavior of odd faults. Time-stamps are particularly useful to measure *time-to-repair* for each injection runs that experience *crash & recovery* events.

#### 4. Experiment setup and execution

The first step of a fault injection experiment is the definition of the set of faults to be applied, which was made with comprehensive automation support from the XceptionNT Experiment Definition (ED) module. The ED module takes input from fault configuration data as present in Table 2, as well as workload and target system configuration data. The output of this module is a “database” of experiments, faults and workload configuration information, which is then used by the XceptionNT Experiment Management (EM) module to control and monitor experiment execution. Xception EM, Experiment Definition, and Database modules are shareable among the family of Xception fault injectors [12]. The same experimental evaluation environment, but applied to a different target system is also used in [13].

injection runs. This way a particular failure can be traced back without any doubt to the root fault/error.

The XceptionNT code running on the target system is configured to run at system start, immediately establishing a TCP/IP connection with the EM module, as soon as system boot is complete. If there is data from the previous injection run it is then sent to the EM, which in turn stores it in the database for posterior analysis. The configuration of the next fault to inject is then collected from the database and sent to the fault injector. The later is responsible for starting the target process (the Oracle service, in this particular case) and do whatever is necessary to inject the fault. Just when Oracle is ready to start execution, a notification is sent back to the EM module, which triggers the workload application process.

Whatever happens after this stage depends on the fault/error injected. In the case that the fault is not injected or stays dormant or the error is masked by normal execution, the workload will terminate before the timeout condition, shutting down the DBMS. The termination event is monitored by XceptionNT, which causes its own

termination (the experiment data collected during the injection run is saved to disk), and reboots the target system, starting a new injection run.

Another possible behavior is the timeout of a particular transaction. In these cases, the TPC-HA DBA entity restores the previous (healthy) execution environment, possibly by means of critical operations such as shutdown and startup. If all operations involved still let TPC-HA terminate on time, the end of the story will be much like the above one. Yet another faulty outcome might be Oracle abort prematurely or hang in such a way that the DBA cannot put it back up again. The EM (on timeout expiration) will terminate these latter injection runs.

## 5. Results

Figure 6 gives a general outlook of the results gathered to assess the impact of faults on the target system. Two different perspectives were used; the OS driven classification is based on the return code of the Oracle server process. *Correct* applies to injection runs where the Oracle process ceases execution with normal exit code. The *correct* classification accounts both for situations where no errors were detected, and when errors were masked by Oracle itself or TPC-HA intrinsic recovery action. *Abort* outcomes match the return code 128, which is issued when Oracle aborts due to an internal error condition such as a non-continuable exception. The *hang* condition corresponds to those cases where the Oracle process is frozen and is terminated by the XceptionNT fault injector. Table 4 summarizes the obtained figures.

Table 4. Impact of faults as seen from the OS perspective

Number of Faults	Correct	Abort	Hang
250	217	20	13
100 %	86.8%	8.0 %	5.2 %

In a non-negligible amount of cases - 8% - the DBMS failed to deliver a proper response, hanging in such a way that even last chance DBA actions like shutdown and startup were insufficient to restore service delivery.

Operating system exceptions detected around 37% of the errors induced in the target system, being the most effective error detection mechanism. It should be stressed that some of these detections are recovered by Oracle, which explains the 86,8% of correct terminations. The user/DBA watchdog, in turn, detected 30.0% of the errors. The TPC-HA consistency tests did not detected a single corruption in user data, which is definitely a good mark on the robustness of the target system.

Table 5 summarizes the detection figures. It should be stressed that the detection vales are counted independently of the detection of other mechanisms.

Table 5. Error detection mechanisms

Number of Faults	Exception Handling	User/DBA watchdog	TPC-HA consistency tests
250	92	75	0
100 %	36.8 %	30.0 %	0 %

Figure 7 depicts the recovery time for the injection runs where at least one successful recovery action was performed. It is worthwhile remember that the term recovery is used here in a broad sense, thus including the close session & retry action performed by the DBA or the end user emulators. As Figure 7 points out, from the 41 injection runs that recovered successfully, a large majority - above 80% - did it within half a minute. Only a few

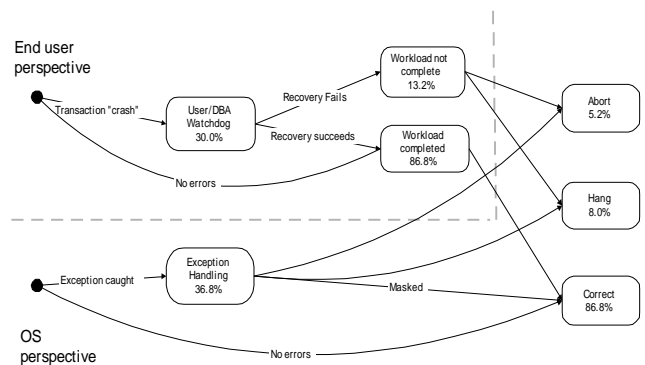


Figure 6. Results overview

recovery actions needed more than a minute repair time, with the top most repair time being around 4 minutes. For this particular experiment a total timeout of 7 minutes was specified which, at first glance, would explain why there are no recovery times above the 4 minutes boundary. However, a few injection runs made to check this thesis did shown that when a recovery action does not succeed within the timeout interval, further progress is not achieved by letting it execute more time.

In terms of *mean time to repair* (MTTR), as was foreseen above it was found out that, in average and with this particular system configuration, a few tens of seconds were sufficient to bring up the DBMS again after error/failure detection. Table 6 presents the MTTR and its associated standard deviation. It is worth to mention that this experimental evaluation worked with an almost zero-latency DBA response to end user error events, which is something that is at most difficult, if not impossible, to meet in real life situations. Furthermore, the load in terms of number of transactions per second is small and Oracle checkpoints are taken very often which favors a fast recovery.

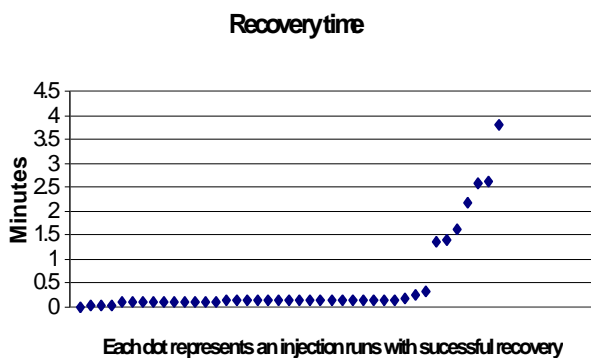


Figure 7. Recovery time from the end user perspective

Table 6. Mean time to repair

% of successful recovery actions (over total injection runs )	MTTR	Standard deviation
16.4 %	29.1 seconds	52.2 seconds

## 6. Conclusions and Future work

An experimental evaluation was performed on the behavior, in presence of transient faults, of a COTS DBMS (Oracle 7.3) installation running on a client/server configuration on a COTS Wintel platform. It was found out that on 8% of the injection runs the DBMS system hang, while above 86% of the times it had resumed execution normally. On the remaining runs (around 5%) the DBMS itself aborted execution.

Most of the error detection score came from OS level exception handling. A detailed analysis revealed that the majority of the faults that were not masked out, caused the DBMS code to produce invalid memory access conditions.

The rate of application recovery success of 55%, measured as the percentage of successful recovery actions over total number of crashes detected, shows that there is plenty of opportunities to improve end to end availability on a client/server database centered computing environment.

Another concluding remark is that end to end time-to-repair, i.e., the average elapsed time from an end user experiencing a failure until service is reestablished, was found to be quite reasonable. Only a small amount of recovery actions needed more than one minute, with the large majority under the 30 seconds boundary.

Last but not least, not even one data integrity violation was detected by the consistency tests. Future work should add more metrics on data integrity aspects, to confirm this result. In fact, user data is only a small amount of the total DBMS data and the integrity of remaining data (metadata) is also essential for delivering proper end user service.

This paper makes several key contributions. First, and to best of our knowledge, it brings up the first fault-injection

results targeted at a real commercial DBMS. The MTTR figures here revealed can be used, combined with fitting MTTF values, to estimate the availability factor due to transient faults on total system availability. Second, it shows that a significant amount of transaction crashes – 34 over 75 total crashes detected – could not be properly handled, i.e., transaction submission did not resumed within an reasonable time either due to DBMS hanging or premature termination. This is a serious drawback for applications that need stringent availability requirements, since this kind of failures will probably need a system reboot in order to reestablish DBMS service delivery. While the results presented here are somehow preliminary, one can say that, at first glance a total COTS DBMS solution seems to be robust enough in presence of transient faults; this holds at least for environments that can tolerate system unavailability while preserving data integrity.

Future work will be targeted at getting a deep insight into the failure models of COTS DBMS, as the basis to establish and validate through fault-injection a tunable framework for increased performability and reliability in COTS DBMS.

## 7. References

- [1] M. T. Ozsú and P. Valduriez, "Distributed Databases Systems: Where are we now?," IEEE Computer, vol. 24, 1991.
- [2] M. P. Sullivan, "System Support for Software Fault Tolerance in Highly Available Database Management Systems," Ph.D. Thesis, 1992.
- [3] W. T. Ng and P. M. Chen. "Integrating Reliable Memory in Databases. In Proceedings of the 1997 Intl. Conf. on Very Large Databases (VLDB), pages 76-85, August 1997.
- [4] S. Chandra and Peter M.Chen, "How Fail-Stop are Faulty Programs?," 28th International Symposium on Fault-Tolerant Computing (FTCS), Munich, Germany, 1998, pp. 240-249.
- [5] Transaction Processing Performance Consortium, "TPC Benchmark A, Standard Specification, Revision 2.0," 1994.
- [6] J.Carreira, H.Madeira, J.Silva, "Xception: A technique for the evaluation of dependability in modern Computers", IEEE Transactions on Software Engineering, Vol.24, No.2, pp. 125-136, February 1998.
- [7] M. Stonebraker, "The Design of the POSTGRES Storage System," presented at VLDB'87, Brighton, England, Morgan Kaufmann, 1987, pp. 289-300.
- [8] "Oracle 7 Server Concepts Manual", Oracle Corporation, 1992.
- [9] D. Costa, J. P. Pereira and H. Madeira, "TPC-HA: Software Requirements for an High-Availability Benchmark", Tech. report, Depto. de Eng. Informática, Universidade de Coimbra, 1998.
- [10] J. Christmansson and R. Chillarege, "Generation of an Error Set that Emulates Software Faults Based on Field Data," presented at FTCS'26, Sendai, Japan, 1996, pp. 304-313.
- [11] A. Wood, "Predicting Client/Server Availability," IEEE Computer, vol. 28, pp. 41-48, 1995.
- [12] Critical Software, "Xception Experiment Manager Environment-Requirements Definition and Analysis", Ref. CRITICAL-1999-REQ-0022-07, Coimbra, 1999.
- [13] João Cunha, Mário Rela and João Gabriel Silva, "Can Software Implemented Fault-Injection be used on Real-Time Systems?," to be presented at the 3rd European Dependable Computing Conference (EDCC-3), Prague, Czech Republic, September 1999.