Recovery and Performance Balance of a COTS DBMS in the Presence of Operator Faults

Marco Vieira
ISEC
Polytechnic Institute of Coimbra
3031 Coimbra - Portugal
mvieira@isec.pt

Henrique Madeira
DEI-FCTUC
University of Coimbra
3030 Coimbra - Portugal
henrique@dei.uc.pt

Abstract

A major cause of failures in large database management systems (DBMS) is operator faults. Although most of the complex DBMS have comprehensive recovery mechanisms, the effectiveness of these mechanisms is difficult to characterize. On the other hand, the tuning of a large database is very complex and database administrators tend to concentrate on performance tuning and disregard the recovery mechanisms. Above all, database administrators seldom have feedback on how good a given configuration is concerning recovery. This paper proposes an experimental approach to characterize both the performance and the recoverability in DBMS. Our approach is presented through a concrete example of benchmarking the performance and recovery of an Oracle DBMS running the standard TPC-C benchmark, extended to include two new elements: a faultload based on operator faults and measures related to recoverability. A classification of operator faults in DBMS is proposed. The paper ends with the discussion of the results and the proposal of guidelines to help database administrators in finding the balance between performance and recovery tuning.

1. Introduction

The ascendance of networked information in our economy and daily lives has increased awareness of the importance of dependability features. In many cases, such as in e-commerce systems, computer outages can result in a huge loss of money or in an unaffordable loss of prestige for companies. In fact, due to the impressive growth of the Internet, some minutes of downtime in a server somewhere may be directly exposed as loss of service to thousands of users around the world.

Databases play a central role in this information infrastructure and it is well known that Database Management Systems (DBMS) have a long tradition in high dependability, particularly in what concerns data integrity and availability aspects. Several basic mechanisms needed to achieve data recovery, such as transactions, checkpointing, logging, and replica control management have been proposed/consolidated in the database arena. However, and in spite of the very important role played by these mechanisms in DBMS, there is no practical way to benchmark their effectiveness or at least to characterize the innumerous configuration alternatives available in typical DBMS products in what concerns the impact of these configurations on database performance and recovery.

The tuning of a large commercial database is a very complex task and database administrators tend to concentrate on performance tuning and often disregard 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.

Database industry holds a reputed infrastructure for performance evaluation and the set of benchmarks managed by the Transaction Processing Performance Council (TPC) are recognized as one of the most successful benchmark initiatives of the overall computer industry. However, data recovery has been largely absent from TPC benchmarking effort. Existing TPC benchmark specify that data recovery features of the database must ensure that data can be recovered from any point in time during the benchmark running, but the benchmarks specifications do not include any procedure to confirm that these mechanisms are working properly or to measure the impact of the recovery mechanisms on the system performance.

The major problem of having pure performance benchmarks (i.e., benchmarks that only measure raw performance) for DBMS and transactional systems in general is that the benchmark results tend to portrait rather artificial situations, as data recovery mechanisms are configured for the minimum impact on transaction performance and the effectiveness of data recovery is totally disregarded. The tight dependence between performance and recovery tuning in modern DBMS urge the definition of practical methods to characterize a given

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.

configuration or compare different DBMS products in a more realistic scenario. Above all, it is important to include in the benchmarks new measures that show the benefit of including better recovery mechanisms in the system or configuring the available mechanisms to achieve the best recoverability.

This paper proposes an experimental approach to characterize both the performance and the recoverability in DBMS by extending existing performance benchmarks to include a faultload (i.e., a set of faults or stressful conditions that activate the recovery mechanisms) and measures related to recoverability. The approach is presented through a concrete example of benchmarking the performance and recovery of an Oracle DBMS running the standard Transaction Processing Performance Council (TPC) TPC-C benchmark [1], extended with two new elements: 1) measures related to recovery time, data integrity violations, and lost transactions and 2) a set of operator faults as a faultload. This experimental approach is generic in the sense that it can be applied to any DBMS (i.e., it has the same field of application as the TPC-C benchmark).

In addition to the first proposal of the extension of TPC-C benchmark to characterize both the performance and the recoverability in DBMS, this paper also has the following contributions:

- Evaluation of the Oracle recovery mechanisms in the presence of operator faults. It is worth noting that it is generally assumed that typical DBMS recovery mechanisms are quite effective, which is in general corroborated by years of intensive use of DBMS in the field. Very few works have evaluated experimentally the behavior of DBMS in the presence of faults. However, all the experimental evaluation works known in the literature have shown that a non-negligible number of faults are not handled correctly by DBMS [2, 3, 4, 5, 6]. While previous papers are "classic" fault injection works, that have injected hardware and software faults, our work is, to the best of our knowledge, the first experimental evaluation of a DBMS with operator faults.
- Proposal of a general classification for operator faults in DBMS. The instantiation of this classification for the Oracle DBMS is presented and a set of tools have been designed and built to cause these faults in the target system in the context of the extension of the TPC-C benchmark. The method used actually inserts typical operator faults by mimicking wrong operator commands using exactly the same means used by the real database administrator in the field, which assures a correct reproduction of operator faults. This approach is generic (i.e., can be applied to any DBMS) and is fully automatic.
- As an extension of an existing performance benchmark, this work is a first contribution towards the proposal of standard dependability benchmarks for DBMS. The concept of dependability benchmarking has gained ground in the last few years and is currently the subject of intense research [7, 8, 9]. The idea is to devise standardized ways to evaluate both dependability and performance of computer systems or components. Comparing to well-established performance benchmarks, dependability benchmarks have two new elements, which are the measures related to dependability and the faultload. In our work, the dependability measures are focused on recoverability (which is directly related to DBMS down time) and data integrity, as these are the most relevant measures in typical database applications, and the faultload consists of operator faults, which is one unanimously considered as a very important source of failures in databases.

The structure of the paper is as follows: the next section presents background on DBMS, especially in what concerns recovery and performance tuning. Section 3 discusses the problem of operator faults in DBMS and presents a classification for this kind of faults based on interviews with database administrators of real databases installations, as well as analysis of typical database administration operations. A short description of the set of tools built to insert these faults is also presented in section 3. The experimental setup is presented in section 4 and the results are presented and discussed in section 5. Section 6 concludes the paper.

2. Background on DBMS

A database is a collection of data describing the activities of one or more related organizations [10][1]. The software designed to assist in maintaining and using databases is called database management system, or DBMS. A DBMS allows users to define the data to be stored in terms of a data model, which is a collection of high-level metadata that hide many low-level storage details. Most DBMS today are based on the relational data model, which was proposed by E. F. Codd in 1970 [11, 12]. The relational data model is very simple and elegant, and defines a database as a collection of one or more relations, where each relation is a table with rows and columns. DBMS based on the relational data model are frequently called relational database management systems. In the rest of the paper we will use the term DBMS to refer to relational database management systems.

In practice, a typical database application (e.g., banking, insurance companies, telecommunications, etc) is a client-server system (either a traditional client-server or a three tier system) where a number of users are connected

[1] This first paragraph is a condensed view of key definitions presented in chapter 1 of the book "Database Management Systems", by R. Ramakrishnan, second edition, McGraw Hill, ISBN 0-07-232206-3.

to a database server via a terminal or a desktop computer (today the trend is to access database servers through the internet using a browser). The user's actions are translated into SQL commands (Structured Query Language: the relational language used by DBMS [13]) by the client application and sent to the database server. The results are sent back to the client to be displayed in the adequate format by the client application.

A very important notion in DBMS is the concept of transaction [14]. In a simplified view, a transaction is a set of commands that perform a given action and take the database from a consistent state to another consistent state. Transaction management is an important functionality of modern DBMS and it is directly related to dependability aspects, particularly in what concerns concurrency control and recovery. Concurrency control is the activity of coordinating the actions of processes that operate in parallel and access shared data, and therefore potentially interfere with each other. Recovery assures that faults (either hardware, software, or operator faults) do not corrupt persistent data stored in the database tables.

In order to correctly deal with concurrency control and recovery, DBMS transactions must fulfill the following properties: **atomicity** (either all actions in the transaction are executed or none are), **consistency** (execution of transaction results in consistent database states), **isolation** (the effects of a transaction must be understood without considering other concurrently executing transactions), and **durability** (the effects of a transaction that has been successfully completed must persist, even when the system has a failure after the transaction is finished). These properties are known as the ACID properties.

The Oracle™ DBMS is one of the leading databases in the market and as one of the most complete and complex database it represents all the sophisticated relational DBMS available today very well. For that reason we have chosen the Oracle 8i DBMS as case study to show the experimental approach proposed to characterize both the performance and the recoverability in DBMS. In the remaining of this section we briefly describe the key features of the Oracle 8i DBMS, with particular emphasis in the recovery and performance tuning and administration.

2.1. Oracle DBMS

An Oracle server consists of an **Oracle database** and one or more Oracle server instances [15]. An Oracle database has logical structures and physical structures. Because physical and logical structures are separate, the physical data storage can be managed without affecting the logical structures. The main physical structures are the control files, the data files, the redo log files, and the archive log files. The following point summarizes the role of each of these physical structures:

- *Control files*: contain all the basic and vital information about the database, such as the physical location of the other files, configuration parameters, etc.
- *Data files*: are the files where the data (user data, metadata, or any other type of data) is stored. An Oracle database can have as many data files as necessary to fulfill the application needs.
- *Online redo log files*: group of files (in a minimum of two) used to record the redo log entries, which are used during database recovery. These files work in a circular way and when all the redo log files are full the Oracle continues writing and overwrites the previous contents of the files (reuse). Given the importance of the redo log information, the redo log files can be replicated to assure correct recovery even when a redo log file is lost.
- *Archive log files*: these files store the redo entries in a permanent way (typically in a tertiary storage device). The archive log files are optional and, when active, avoid the loss of redo log information due to the circular use of the online redo log files.

The user space (i.e., the space used to store user objects such as tables, indexes, etc) is available through a set of logical structures: tablespaces, segments, extents, and data blocks. The tablespace is a logical area that is physically composed by one or more data files. Users receive quotas in tablespaces and an Oracle database can have as many tablespaces as needed for a good administration of the storage space. Any data object in Oracle (table, index, etc) is associated to one segment and segments acquire space from a tablespace through units called extents. Finally, each extent is composed by a given number of data blocks, which is the basic storage unit.

The combination of the background processes and memory buffers is called an **Oracle instance**. Every time an instance is started, a system global area (SGA) is allocated (area of memory used for database information) and Oracle background processes are started.

An Oracle instance has two types of processes: user processes and Oracle processes. A user process executes the code or commands from application programs. The Oracle processes include server processes that perform work for the user processes and background processes that perform maintenance work for the Oracle server. Some of the most important processes are: the database writer (DBWR), the log writer (LGWR), the checkpoint (CKPT), and the archive writer (ARCH).

During the normal database operation, all the activities are stored as entries in the online redo log files. The entries are cached in the redo log buffer and regularly saved into the log files by the LGWR process (actually, the redo log buffer is written into the log files whenever a user transaction commits). Another important goal of the LGWR is to record the activities as frequently as possible,

allowing the DBWR to delay its data writing operations. It is worth noting that typically a redo log entry is much smaller than the corresponding data operation described in the log entry, which means that it is much faster to write the log entries into the disk than the corresponding data. If the log archive option is on, the archive log process (ARCH) copies online redo log files to a tertiary storage device once a given online redo log file becomes full.

Oracle performs periodical checkpoints, which represent consistent states of the database from which it is possible to start recovery. The recovery consists of a forward phase, in which the redo entries recorded in the redo log are applied to regenerate the data, followed by a backward phase, in which some unrecoverable transactions are rolled back to bring the database to a clean state. To reduce downtime due faults, Oracle uses a basic process pair mechanism. It consists of a stand-by database running in a different machine that is kept in a permanent recovery process by executing the redo logs of the primary database. This way, if the primary database goes down the stand-by database can replace it and resume the work very quickly, greatly reducing the down time.

2.2. Oracle performance and recovery tuning

The performance and recoverability of an Oracle database depend on many system elements, that can be summarized as follows: memory and processes, physical storage, database objects, SQL commands, and recovery mechanisms. Obviously, the logging and checkpointing activities necessary to recover from faults also cause some performance degradation, which is very dependent on specific tuning options. Furthermore, several components of the underlying platform (e.g., hardware, operating system, and network) can also impact the performance and recovery, but these aspects are clearly outside DBMS performance and recovery tuning. The following points summarize the key aspects of Oracle 8i performance and recovery tuning and give an idea of the huge complexity of tuning a large database installation. As general reference for database concepts and tuning see [10] and for a more specific reference concerning tuning of Oracle 8i see [15].

- *Memory and processes*: the goal is to optimize the I/O operations through the definition of the optimal size of the different memory areas in the SGA and the cache and buffer policies. Additionally, it is possible to optimize the number of some processes such as the DBWR or LGWR.
- *Physical storage*: the goal is to minimize contention when accessing database files and fragmentation in the physical storage of database objects. Many aspects affect the physical storage, such as the distribution of the database files by the available disks, and the numerous parameters used to configure the space allocation to database objects through extents and data blocks.
- *Database objects*: the definition of database objects also impacts the performance. For example, the normalization of the tables, the creation of adequate indexes, clustering or partitioning of some objects, etc.
- *SQL commands*: in addition to the DBMS parameters related to the query optimization and SQL execution, the transaction design (i.e., set of SQL commands) also affects performance. In this last case, the goal is to minimize the performance impact of concurrency control by avoiding transaction contention due to object blocking. The optimization of SQL query execution is mainly related to the policy used by the DBMS query optimization module (cost or rule based).
- *Recovery mechanisms*: the tuning of the recovery mechanism is clearly the most difficult part when a well-balanced tuning is desired. The goal is to conciliate aspects such as the minimization of recovery time and the impact of faults in terms of lost transactions with maximum performance. Some examples of parameters related to recovery mechanisms are size and number of online redo log files, archive log option, checkpointing policy, and stand-by database.

It is worth noting that database tuning must also take into account the interdependencies among most of the elements mentioned above. In practice, in a complex DBMS such as Oracle 8i the performance and recovery tuning may require the definition of thousands of parameters, which clearly show the difficulties faced by database administrators in handling this problem. This emphasizes the need for comprehensive benchmarks that include both performance and recoverability measures. In this sense, the present work is a first step towards this direction.

3. Operator Faults in DBMS

Our proposal of extending the TPC-C performance benchmark to measure both performance and recoverability requires the introduction of two new elements: recoverability measures and faultload. The representativeness of these new elements is, obviously, of utmost importance. Concerning the faultload the challenge is to define a representative set of faults and find practical and consistent ways to introduce these faults in the system under test (to make it possible to repeat the experiments or to port them to other systems). In the same way we have concentrated on measures of recovery time and integrity violations, instead of more general dependability measures (because these measures are the most relevant to databases and are directly related to database availability and data integrity), we also decided to focus on operator

faults, as these faults are unanimously considered by the database community as responsible for most of the failures in database systems.

Studies on the underlying causes of database failures in the field are not generally available to the public, as the organizations prefer not to disclose this information. Most of the published studies are not directly focused on database systems. Nevertheless, published studies clearly point out software faults and operator faults as the most frequent causes for computer failures [16, 17, 18, 19, 20]. Furthermore, the great complexity of database administration tasks and the need of tuning and administration in a daily basis, clearly explain why operator faults (i.e., wrong database administrator actions) are prevalent in database systems. Several interviews with database administrators of real databases installations conducted on behalf of this work to define a classification for operator faults have also confirm the prevalence of operator faults in database systems.

Although software faults are considered an important source of failures, the emulation of software faults is still a research issue and there are no practical tools readily available to inject this kind of faults [21, 22]. Thus, we decided to address only operator faults in this first study.

Operator faults in database systems are database administrator mistakes. End-user errors are not considered, as the end-user actions do not affect directly the dependability of DBMS. In fact, the end-users do not have direct access to the DBMS (e.g., do not execute SQL commands even in user accounts), and they are only allowed to use the database through well-defined interface applications that isolate them from the DBMS.

The list of tasks performed by a database administrator is very long. Thus, we have decided to analyze the database administrator tasks and group them in classes related to the major groups of database administration operations. This analysis was based on the study of core functions available in all DBMS [10], interviews with database administrators of real databases installations, and examination of database scripts used to help some database administration tasks. Table 1 shows the proposed classes of DBMS operator faults (each class of faults corresponds to many types of mistakes) and gives some examples of faults for each class.

These major groups of operations can be found in any commercial DBMS, as they are related to core functions available in all DBMS [10]. Although the details of some administrator operations are specific to each DBMS, the standard SQL used by the vast majority of DBMS greatly simplifies the establishment of equivalent administrator mistakes in different DBMS implementation. Some operator faults, however, are intimately related to specific features of a given DBMS and those ones may have no counterparts in other DBMS.

As we can see from the groups of operations used to define each fault class in Table 1, database administrators manage all aspects of DBMS. Obviously, administrator mistakes easily damage DBMS availability and performance, which shows the interest of benchmarking the behavior of DBMS in the presence of these faults. 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 (i.e., we do not emulate faults as it is usual in traditional fault injection: we really reproduce operator faults). In order to make the procedure fully automatic, faults can be injected by a set of scripts that perform the wrong operation at a given moment (the fault trigger). As usually happens in traditional fault injection, the fault trigger can be defined in such a way that operator faults can be uniformly distributed over time or can be synchronized with a specific event or command of the workload.

Classes	Description
Memory & processes admin.	Mistakes in the administration of processes and memory structures. Incorrectly define the memory allocation and processes initialization parameters are typical faults related to processes and memory administration. Another typical fault is the accidental database shutdown that causes the loss of service.
Security manag.	Mistakes in the attribution of passwords, access privileges, and disk space to users. These are very problematic faults in database administration, as their effects are difficult to detect.
Storage admin.	Mistakes in the administration of the physical and logical storage structures. Common examples of this class of faults are: the removal or corruption of database files, the incorrect distribution of files by several disks, and letting the storage structures run out of space.
Database object admin.	Errors related to the management of the user objects. The removal of a user object (e.g., table, index, etc.), the incorrect configuration of the user objects, and the incorrect use of the optimization structures are common faults related to the database schema administration.
Recovery mechan. admin.	Mistakes in the configuration and administration of the database recovery mechanisms. Some typical examples are: the inexistence of backups, the removal or corruption of a log file, and the inexistence of archive logs.

Table 1. Classes of DBMS operator faults.

3.1. Example of operator faults in Oracle DBMS

Table 2 shows an instantiation of the operator fault classes proposed in Table 1 for the case of Oracle 8i DBMS, where each line represents specific types of operator faults. The right column indicates whether each type of faults is likely be found in other DBMS or not, based on the analysis of fault characteristics and their relation with general architectural features of DBMS [10].

A simplified comparative analysis of actual implementations of DBMS for some major vendors (Oracle, Microsoft, and Sybase) has also been done. The notation used is the following: "Yes" means that exactly the same type of fault can be found in other DBMS; "Equivalent" means that the type of fault needs to be translated to take into account implementation details but a fault with equivalent effects can be found in other DBMS; "Oracle" means that the type of fault is specific of Oracle 8i DBMS. As we can see, most of the faults are expected to be found in other DBMS. Although a more comprehensive study is required to evaluate the portability of these types of faults across different DBMS implementations, this simplified analysis adds a flavor of generality to the data presented in Table 2, and suggests that a faultload based on a subset of these types of operator faults could be fairly general.

	Type of operator fault	Other
Memory & processes admin.	Making a database instance shutdown	Yes
	Removing or corrupting the initialization file	Yes
	Incorrect configuration of the SGA parameters	Yes
	Incorrect config. max. number of user sessions	Yes
	Killing a user session	Yes
Security managem.	Database access level faults (passwords)	Yes
	Incorrect attrib. of system and object privileges	Equivalent
	Attribution of incorrect disk quotas to users	Equivalent
	Attribution of incorrect profiles to users	Equivalent
	Incorrect attribution of tablespaces to users	Oracle
Storage administration	Delete a controlfile, tablespace or rollback seg.	Oracle
	Delete a datafile	Equivalent
	Incorrect distribution of datafiles through disks	Yes
	Insufficient number of rollback segments	Oracle
	Set a tablespace offline	Oracle
	Set a datafile offline	Equivalent
	Set a rollback segment offline	Oracle
	Allow a tablespace to run out of space	Oracle
	Allow a rollback segment to run out of space	Oracle
Database object admin.	Delete a database user	Yes
	Delete any user's database object	Yes
	Incorrect config. object's storage parameters	Equivalent
	Set the NOLOGGING option in tables	Oracle
	Incorrect use of optimization structures	Yes
Recovery mechanisms administration	Delete a redo log file or group	Equivalent
	Store all redo log group members in same disk	Equivalent
	Insufficient redo log groups to support archive	Equivalent
	Inexistence of archive logs	Equivalent
	Delete a archive log file	Equivalent
	Store archive files in the same disk as data files	Equivalent
	Backups missing to allow recovery	Equivalent

Table 2. Example of concrete types of DBMS operator faults for Oracle 8i.

3.2. Operator faults emulation and recovery

As mentioned before, the injection of operator faults in a DBMS can be easily achieved by reproducing common database administrator mistakes. However, in order to emulate an operator fault in a way similar to what happens in real world the set of steps represented in Figure 1 must be followed. A very important aspect concerning the fault emulation is the instant of activation of the faults (fault trigger). The same fault activated in different moments may cause different behaviors according to the system state, which means that different instants for the injection of faults must be chosen.

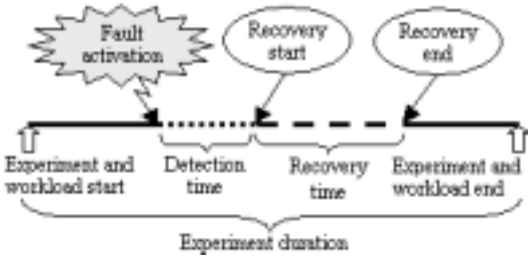

Figure 1. Steps in the injection of an operator fault.

Another relevant aspect to automate the experiments is that it is necessary to evaluate the type of recovery procedure that may be required after each fault. This means that the scripts used to inject each type of fault must also include all the steps required to start the adequate recovery procedure, which is much more complex than the actual reproduction of the operator faults. The starting instant of the recovery depends on the time needed to detect the error. Due to the fact that in real situations that time is highly human-dependent, a typical detection time (for experiment purposes) has to be established for each type of operator fault.

4. Experimental setup

The basic platform used in the experiments presented in this paper consists of two Intel Pentium III servers with 256MB of memory, four 20GB hard disks, running the Windows 2000 operating system and connected through a dedicated fast-Ethernet network.

As mentioned earlier in the paper, the TPC-C performance benchmark [1] running on top of Oracle 8i database server is used as case study for this joint evaluation of performance and recoverability. The TPC-C performance benchmark represents a typical database installation. This workload includes a mixture of read-only and update intensive transactions that simulate the activities found in many complex OLTP application environments. The performance metric for this benchmark is expressed in transactions-per-minute-C (tpmC).

Figure 2 shows the key components of the experimental setup. The two basic configurations used in the experiments consist of a single database server (which is the standard configuration) and a configuration with a stand-by database to study the benefits of using a spare server to speed up recovery. The TPC-C specification also includes an external driver system that emulates all the client applications and respective users during the benchmark run. This driver system has been extended to

handle the insertion of the operator faults. Additionally, the driver system also records the base data needed to get the recovery and integrity measures, which consist of **recovery time**, **lost transactions**, and detection of **integrity violations**. It is worth noting that these measures are taken from the end-user point of view, which means that, for example, the recovery time includes all the time needed to recover the Oracle server plus the time needed to reestablish the transaction execution at the client application level (i.e., as it would be seen by the end-user).

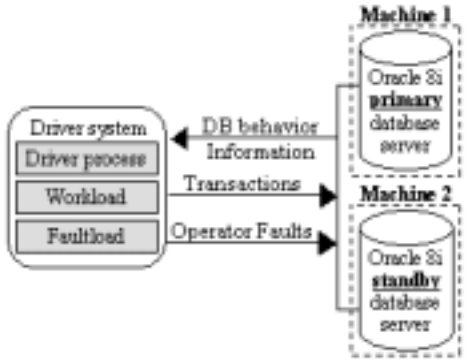

Figure 2. Experimental setup layout.

The operator faults are injected through a set of scripts and following the steps represented in Figure 1. In order to make it easy to reproduce the experiments we have decided to inject the faults in three specific moments (instead of using a random distribution for the fault triggers), respectively after 150, 300, and 600 seconds from the TPC-C start. The faults injected in the first instant (150 seconds) affect the system when it is speeding up to reach the nominal transaction performance. The faults injected in the second instant (300 seconds) affect the system when it is on its maximum processing throughput. Finally, the faults injected in the third instant (600 seconds) affect the system when a large amount of transactions has already been processed. The duration of each individual experiment was 20 minutes and the execution of all the experiments is fully automatic.

Considering the types of faults presented in Table 2, we have excluded from this first study the classes of faults "security management", because the security issues are out of scope of this work, and "recovery mechanisms administration", as most of these faults just affect database performance and would require two consecutive faults to affect the system in other visible ways. In fact, after a first fault affecting the recovery mechanisms we would need a second fault of other type to activate the recovery and reveal the effects of the first fault.

From the three operator faults classes considered ("memory and processes administration", "storage administration", and "database object administration" faults) we have selected six main types of fault. These faults have been chosen based on their ability to emulate the effects of other types of faults (to minimize the number of experiments needed), on the diversity of impact in the system, and on the complexity of required recovery. The basic six types of faults include: Shutdown abort, Delete a datafile, Delete a tablespace, Set a datafile offline, Set a tablespace offline, and Delete user's object.

A total of 146 faults have been injected from the six types of faults mentioned above, covering all the tested configurations. The scripts to emulate the operator faults were programmed in Perl and SQL and can easily be ported to other DBMS.

5. Experimental results and discussion

Three sets of experiments have been conducted. The first set of experiments evaluates the effectiveness of different configurations of the basic recovery mechanism (online redo logs). The second set of experiments evaluates different configurations of the archive log mechanism. The last set of experiments assesses the effectiveness of the stand-by database mechanism. The following sub-sections present and discuss these results.

5.1. Results with basic recovery mechanism

When the online redo log mechanism is used alone it only guarantees the recovery of Shutdown Abort type of faults, due to the redo log files reuse. However, some other faults can be recovered provided that the last database backup was made after the last reuse of any redo log file.

The frequency of checkpoints is one important recovery factor affecting the database performance. Obviously, very frequent checkpoints tend to reduce the recovery time. Other relevant factor to reduce the recovery time is to minimize the time a given data block marked as dirty remains in the database cache (because this reduces the amount of data to be recovered).

The checkpoint frequency and database cache consistency algorithm are dependent on several parameters related to recovery mechanisms, such as the redo log file size, the number of redo log groups, and the checkpoint timeout (controlled by the log_checkpoint_timeout parameter). The actual trigger of a checkpoint and the cache activity profile are also intimately related to the transaction activity (dependent, in turn, on the user applications), which is responsible for the difficulties in forecasting the impact of recovery configurations on performance (that is why we need benchmarking).

In order to assess the impact of different checkpointing policies on performance and recovery, we have defined a set of different recovery configurations and we have injected the faultload (faults represented by the Shutdown Abort type) to measure the recovery time. Table 3 shows the set of configurations and Figure 4 shows the results.

Results show a clear impact of the different recovery configurations on database performance. However, we have observed a clear impact on the performance only for configurations having high checkpointing rates. As expected, the recovery time for these configurations is the lowest. An important conclusion from Figure 4 is that we can increase the checkpoint rate, reducing the recovery time, without causing a severe impact on performance.

Config.	File Size	Redo Log Groups	Checkpoint Timeout	# CKPT per Experiment
F400G3T20	400 MB	3	1200 sec.	1
F400G3T10	400 MB	3	600 sec.	1
F400G3T5	400 MB	3	300 sec.	1
F400G3T1	400 MB	3	60 sec.	1
F100G3T20	100 MB	3	1200 sec.	5
F100G3T10	100 MB	3	600 sec.	5
F100G3T5	100 MB	3	300 sec.	5
F100G3T1	100 MB	3	60 sec.	4
F40G3T10	40 MB	3	600 sec.	13
F40G3T5	40 MB	3	300 sec.	12
F40G3T1	40 MB	3	60 sec.	14
F10G3T5	10 MB	3	300 sec.	54
F10G3T1	10 MB	3	60 sec.	55
F1G6T1	1 MB	6	60 sec.	319
F1G3T1	1 MB	3	60 sec.	380
F1G2T1	1 MB	2	60 sec.	263

Table 3. Set of recovery configurations used.

The configurations F400G3T1 and F100G3T1, in spite of having a low checkpointing frequency, have a very short recovery time. This is due to the short checkpoint timeout (60 seconds) that reduces the amount of data to be recovered due to the frequent writes of dirty cache data blocks into database files.

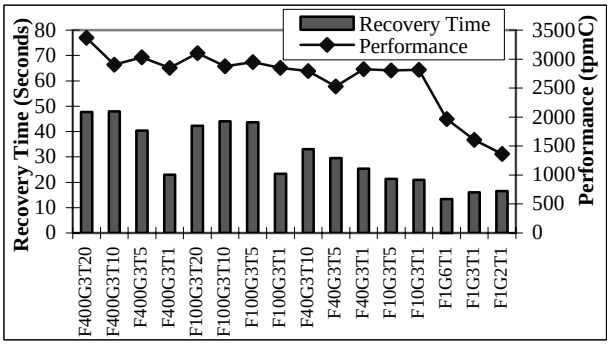

Figure 4. Performance vs recovery time.

One very important conclusion is that all the faults represented by the Shutdown Abort have not caused data integrity violations or loss of committed transactions. Only the transactions under execution when the fault is injected have to be rolled back, but in this case the end-user is notified that the transaction has been aborted.

5.2. Results with Archive Logs

The activation of the archive logs is extremely important because the system can recover from most of the operator faults, as the archive logs store all the redo sequence. In this second set of experiments we activated the archive logs to evaluate their impact on performance and recoverability. These experiments used the configurations G40G3T10 to F1G2T1 (the other configurations are not relevant because the large size of the online redo log files would require a longer experiment time to start the log entries archiving) presented in Table 3 and include the injection of all types of operator faults presented in section 4.

The results presented on Figure 5 show a moderate impact on database performance, which suggests that the archive log option must always be activated.

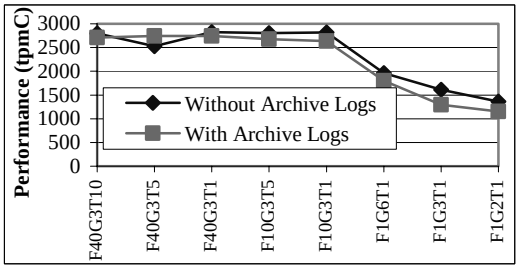

Figure 5. Performance with and without archive logs.

Another interesting aspect is to observe how the recovery time depends on the type of fault and database recovery configurations. The Oracle database has two types of recovery: incomplete and complete recovery (respectively with and without loss of committed transactions). The different types of operator faults are associated to one of these types of recovery. Table 4 presents the recovery times observed for the operator faults that caused incomplete recovery and Table 5 presents the recovery times for the faults that caused complete recovery.

Fault	Configuration	Recovery time (seconds)		
		Injection 150 Sec	Injection 300 Sec	Injection 600 Sec
Delete user's object	F40G3T10	264	284	339
	F40G3T5	235	256	335
	F40G3T1	235	293	342
	F10G3T5	265	277	372
	F10G3T1	260	316	380
	F1G6T1	340	474	>600
	F1G3T1	301	384	>600
	F1G2T1	300	343	>600
Delete tablespace	F40G3T10	277	304	357
	F40G3T5	277	309	360
	F40G3T1	282	291	365
	F10G3T5	278	287	382
	F10G3T1	282	315	364
	F1G6T1	353	462	>600
	F1G3T1	356	472	>600
	F1G2T1	249	423	>600

Table 4.Recovery time for faults with incomplete recov.

Results show that different configurations have different recovery times. Once again, the recovery time depends on the checkpoint frequency, but only for the faults that do not use the archive log files (shutdown abort, set tablespace offline, and set datafile offline).

For the faults that require the processing of the archive log files the recovery time is not influenced by the checkpoints frequency, because the recovery starts from a checkpoint that is not stored in the online redo log files anymore. In practice, the recovery always starts from the reposition of database files from a backup. In this case, the recovery time depends on the size of the archive log files. Small files tend to increase the recovery time because a large amount of files have to be processed. In this case, the best recovery time is obtained for larger files.

Fault	Configuration	Recovery time (seconds)		
		Injection 150 Sec	Injection 300 Sec	Injection 600 Sec
Shutdown abort	F40G3T10	31	34	35
	F40G3T5	28	31	34
	F40G3T1	24	24	28
	F10G3T5	21	21	22
	F10G3T1	19	22	22
	F1G6T1	14	13	13
	F1G3T1	17	13	18
	F1G2T1	18	17	16
Delete datafile	F40G3T10	33	42	57
	F40G3T5	39	40	64
	F40G3T1	32	43	64
	F10G3T5	36	51	76
	F10G3T1	35	55	77
	F1G6T1	59	109	191
	F1G3T1	55	98	139
	F1G2T1	48	70	115
Set datafile offline	F40G3T10	10	11	17
	F40G3T5	9	11	18
	F40G3T1	11	10	11
	F10G3T5	6	4	7
	F10G3T1	6	6	5
	F1G6T1	3	1	5
	F1G3T1	4	1	4
	F1G2T1	6	2	3
Set tablespace offline		Always close to 1 second		

Table 5.Recovery time for faults with complete recov.

A relevant result for the faults that caused incomplete recovery is that some commited transactions have been lost. However, the number of lost commited transactions was constantly very small because the recovery was always started immediately after the fault occurrence. This doesn't happen in the field, because the detection time may be dependent on the database administrator actions. In our experiments we assumed a constant (and small) detection time, as the goal of our work is to assess the effectiveness of the recovery mechanisms and not the database administrator reaction time and capabilities. Another very important conclusion is that none of the operator faults caused data integrity violations.

5.3. Results with Stand-by Database

The main goal of the stand-by database is to reduce the recovery time and, consequently, minimize downtime. The stand-by database is kept in a permanent recovery state in which it processes the redo entries in archive logs of the primary database. For this reason, different configurations of the archive logs and online redo logs cause different behaviors on the stand-by database. Furthermore, in addition to the performance degradation caused by the activation of the archive logs mechanism, stand-by database also requires some mean to share archive log files between both machines, which may also cause some overhead. Lines in Figure 6 show the performance results for both the stand-by database and the archive log mechanisms. As we can see, both the archive logs and the stand-by database cause a moderate performance impact, which suggests that performance penalty is not an excuse for not using these more elaborate Oracle recovery mechanisms.

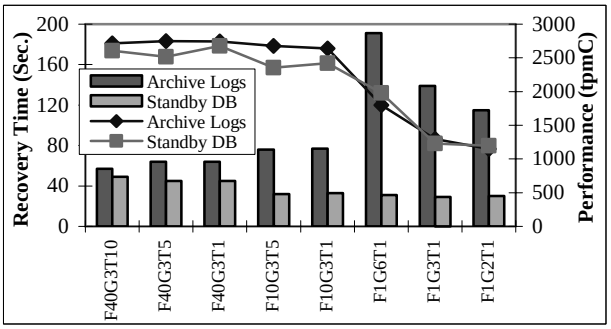

Figure 6. Performance and recovery time with archive logs and stand-by database.

The recovery time in a stand-by database is the same for all the faults. This is due to the fact that the stand-by database activation time is independent of the primary database. Figure 6 shows the recovery times for operator faults injected 600 seconds after the workload starting. In order to allow results comparison, Figure 6 also presents the recovery times obtained in the experiments with the archive log mechanism for the Delete Datafile fault, injected 600 seconds after the workload start. As can be seen, a considerable reduction of the recovery time is achieved.

However, in the stand-by database configuration, if the primary database current redo log group cannot be archived due to the crash of the system, the transactions associated to the log entries saved on that group are lost, and the corresponding commited transactions cannot be recovered. To reduce the number of lost redo log entries, the size of the redo log files must be the as small as possible. Figure 7 shows the results concerning the lost transactions using different redo log files sizes and different number of redo log groups.

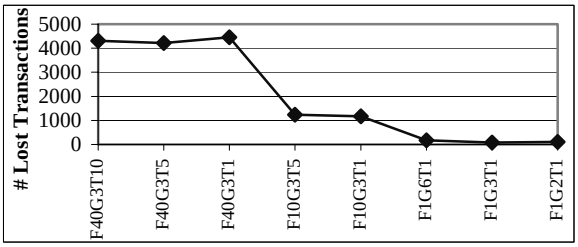

Figure 7. Lost transactions in the stand-by database.

6. Conclusion

This paper proposes an experimental approach to characterize both the performance and the recoverability in DBMS by extending the standard TPC-C benchmark to include two new elements: a faultload based on operator faults and measures related to recoverability. Given the rather artificial performance results achieved by typical performance benchmarks (because normally do not take into account the balance between performance and recoverability) we think that this kind of benchmarks is very useful to characterize DBMS in realistic scenarios. Additionally, the same environment can be used to characterize recovery mechanisms configurations in DBMS, including measures such as recovery time, data integrity violations, and lost transactions.

The paper also proposes a classification of operator faults for DBMS, and defines a comprehensive set of types of operator faults. A set of tools have been designed and built to reproduce operator faults in Oracle DBMS, which is, to the best of our knowledge, the first proposal of an environment to inject operator faults in DBMS.

The experimental results obtained by extending the TPC-C benchmark with our approach were analyzed and discussed in detail. These results clearly show that recovery mechanisms do affect peak performance but, at the same time, show that it is possible to configure the Oracle DBMS to get good recovery features with moderate or even minimal performance impact. In our opinion, it would be difficult to characterize these well-balanced configurations without an experimental approach such as the one proposed in this paper.

7. References

[1] Transaction Processing Performance Consortium, “TPC Benchmark C, Standard Specification, Version 5.0,” 2001, available at: http://www.tpc.org/tpcc/.

[2] S. Bagchi, Y. Liu, K. Whisnant, Z. Kalbarczyk, R. Iyer, Y. Levendel, “A Framework for Database Audit and Control Flow Checking for a Wireless Telephone Network Controller”, Proc. of the 2001 Intl. Conference on Dependable Systems and Networks, Gotheburg, Sweden, 1-4 July, 2001, pp.225-234.

[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, Munich, Germany,1998, pp. 240-249.

[5] M. Sabaratnam, Ø. Torbjørsen and S.Hvasshovd, “Evaluating the Effectiveness of Fault Tolerance in Replicated Database Management Systems”, Proceedings of 29th International Symposium on Fault-Tolerant Computing, June 15-18, Madison, Wisconsin, 1999, pp. 306-313.

[6] D. Costa, T. Rilho, and H. Madeira, “Joint Evaluation of Performance and Robustness of a COTS DBMS Through Fault-Injection”, IEEE/IFIP Dependable Systems and Networks Conference – DSN (FTCS-30 e DCCA-8), New York, USA, 25-28 June, 2000, pp. 251-260.

[7] Aaron Brown and David Patterson, "Towards availability benchmark: a case study of software RAID systems", Proceedings of 2000 USENIX Annual Technical Conference, San Diego, California, USA, June 18-23, 2000, pp 263-276.

[8] H. Madeira and P. Koopman, “Dependability Benchmarking: making choices in an n-dimensional problem space”, First Workshop on Evaluating and Architecting System Dependability (EASY), DSN-2001, Göteborg, Sweden, July 1, 2001.

[9] K. Kanoun, J. Arlat, D. Costa, M. Dal Cin, P. Gil, J-C. Laprie, H. Madeira, and N. Suri, “DBench: Dependability Benchmarking”, in Suppl. of the Int. Conference on Dependable Systems and Networks, DSN-2001, Chalmers University of Technology, Göteborg, Sweden, 2001, pp. D.12-D.15.

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

[11] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks ", Communications of the ACM (1970).

[12] E. F. Codd, The Relational Model for Database Management (Addison-Wesley Publishing Company, 1990), ISBN 0-201-14192-2.

[13] C. J. Date and Hugh Darwen, “The SQL Standard”, Third Edition (Addison-Wesley Publishing Company, 1993), 414 pages; paperbound; ISBN 0-201-55822-X.

[14] J. Gray and A. Reuter, “Transaction Processing: Concepts and Techniques”, The Morgan Kaufmann Series in Data Management Systems, Jim Gray, Series Editor 1993, ISBN 1-55860-190-2.

[15] Oracle Corp., "Oracle 8i Server Concepts Manual", 1999.

[16] 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.

[17] M. Sullivan and R. Chillarege, “Comparison of Software Defects in Database Management Systems and Operating Systems”, Proceedings of the 22nd IEEE Fault Tolerant Computing Symp., FTCS-22, pp. 475-484, July 1992.

[18] 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.

[19] 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.

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

[21] 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.

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