Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Performance Evaluation 56 (2004) 187–212

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

Joint evaluation of recovery and performance of a COTS DBMS in the presence of operator faults

Marco Vieira [a,*], Henrique Madeira [b]

[a] *ISEC/CISUC, Polytechnic Institute of Coimbra, 3031 Coimbra, Portugal*
[b] *DEI/CISUC, University of Coimbra, 3030 Coimbra, Portugal*

Abstract

A major cause of failures in large database management systems (DBMS) is operator/administrator faults. Although most of the complex DBMS available today 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 of 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/administrator faults in DBMS is proposed. A set of tools have been designed and built to reproduce operator faults in an Oracle 8*i* DBMS, using exactly the same means used in the field by the real database administrator. This experimental approach is generic (i.e., can be applied to any DBMS) and is fully automatic. 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.

Keywords: Performance and recoverability; Experimental evaluation; Database management systems; Operator faults

1. Introduction

The ascendance of networked information in our economy and daily lives has increased the 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

* Corresponding author.
E-mail addresses: mvieira@isec.pt (M. Vieira), henrique@dei.uc.pt (H. Madeira).

0166-5316/$ – see front matter © 2003 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2003.07.006

to 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 to the impact 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 evaluation has been largely absent of TPC benchmarking effort. Existing TPC benchmarks specify that data recovery features of the database must ensure that data can be recovered from any point of time during the benchmark run, but the benchmark 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 measures 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 urges the definition of practical methods to characterize a given 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 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 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 performance and 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 [2–6]. 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 three of the leading DBMS in the market (Oracle 8*i*, Sybase Adaptive Server 12.5, and Informix Dynamic Server 9.3) is presented. A set of tools have been designed and built to cause these faults in the Oracle 8*i* DBMS 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 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 gaining ground in the last few years and is currently subject of intense research [7–9]. 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 are directly related to DBMS downtime) and data integrity, as these are the most relevant measures in typical database applications, and the faultload consist of operator faults, which are unanimously considered as a very important (if not the most frequent) source of failures in databases.

The structure of the paper is as follows: the next section presents some 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. The experimental setup and a short description of the set of tools built to insert these faults in an Oracle 8*i* DBMS are presented in Section 4. The results are presented and discussed in Section 5 and Section 6 summarizes the most relevant configuration scenarios. Section 7 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. The need for such systems, as well as their use, has grown rapidly in the last two decades. Commercially, DBMS represents one of the largest and most dynamic market segments. 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 Codd [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, all sort of traveling businesses, telecommunications, wholesale retail, complex manufacturing processes, etc.) is a client/server

[1] This section is a condensed view of key definitions presented in Chapter 1 of [10].

system (either a traditional client/server or a three tier system) where a number of users are connected 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 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 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 takes 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 to concurrency control (essential to assure data integrity) and recovery. Concurrency control is the activity of coordinating the actions of processes that operate in parallel and access shared data, and therefore potentially interferes 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 transaction finishing). 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 very well the typical sophisticated relational DBMS available today. For that reason we have chosen the Oracle 8*i* DBMS as case study to demonstrate 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 8*i* DBMS, with particular emphasis to what concerns to 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 (see Fig. 1). 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. Usually, the database administrator configures the Oracle to keep the control files replicated, to assure that at least one replica survives disk failures or operator faults that corrupt the control files.
- *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 (detailed further on). These files work in a circular way and when all the redo log files are full the Oracle continues writing redo log entries and overwrites the previous contents of the files (redo log file reuse). Given the importance of the redo log information, the database

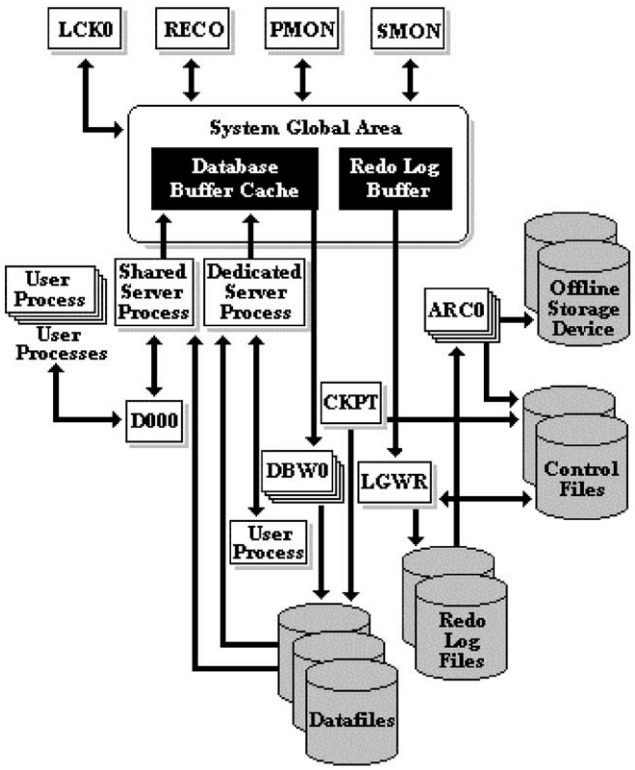

Fig. 1. The Oracle 8*i* background processes and recovery structures [15].

administrator can configure the Oracle to keep the redo log files replicated to assure correct recovery, even when one the redo log files of one group (a set of replicas is called a redo log group) are 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 (new files can be added to a tablespace whenever needed). 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 normally one segment contains only one object) and segments acquire space from a tablespace through units called extents. Finally, each extent is composed by a given number of data blocks (depending from the segment configuration), which is the basic storage unit in Oracle.

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 and Oracle background processes are started. The system global area is an area of memory used for database information and includes two important memory structures: the database buffer cache and the redo log buffer (see Fig. 1).

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). The following subsection briefly explains how these processes work.

2.1.1. Recovery in Oracle

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). One important goal of the LGWR (in addition to its main goal of creating the log) is to record the activities as frequent 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 disk than the corresponding data. In other words, the creation of the log required by recovery is also used to optimize the Oracle data cache policy and reduce the impact of logging in the performance. If the log archive option is on, the archive log process (ARCH) will copy online redo log files to a tertiary storage device every time one online redo log file becomes full.

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 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 standby 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 standby database can replace it and resume the work very quickly, greatly reducing the downtime.

2.1.2. Oracle performance and recovery tuning

The performance and recoverability of an Oracle database depend on many system elements, which 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 cause also some performance degradation, which is very dependent on specify tuning options. Furthermore, several components of the underlying platform 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 8*i* performance and recovery tuning and give an idea of the huge complexity of the 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 8*i*, see [15].

- *Memory and processes*: The goal is to optimize the I/O operations through the definition of 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 replication of some types of files (control and online redo log files), the distribution of the database

files by the available disks, configuration of the tablespaces, 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, hashing, 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 the object blocking. The optimization of SQL query execution is mainly related to the policy used by the DBMS query optimization module (based on cost or based on rules).
- *Recovery mechanisms*: The tuning of the recovery mechanism is clearly the most difficult part when a well-balanced tuning is desired (in opposition to a rather artificial configuration that only favors performance). 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 the size and number of online redo log files, archive log options, checkpointing policy, and standby 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 8*i* the performance and recovery tuning may require the definition of hundreds of parameters, which clearly show the difficulties faced by database administrator 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 another systems). In the same way we 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 for many reasons easy to understand. Most of the published studies are not directly focused to database systems. Nevertheless, published studies clearly point out software faults and operator faults as the most frequent causes for computer failures [16–21]. Furthermore, the great complexity of database administration tasks and the need of tuning and administration in a daily basis, clearly explains why operator/administrator faults 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 (detailed further on) have also confirmed the prevalence of operator faults in database systems.

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 interactive SQL commands even in user accounts), and they are only allowed to use the database through well-defined interface applications that isolate them form the DBMS (e.g., an ATM interface allows the users to perform high-level operations such as withdraw money or check account balance, and there is no room for user mistakes that affect the system dependability).

A database administrator (DBA) manages 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 (and this picture is not likely to change in the near future). Obviously, administrator mistakes easily hurt DBMS availability, 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. That is, 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.

3.1. Main DBMS administration areas

The DBA is responsible for developing and maintaining a database installation. Typically, a DBA is someone with years of practice and has a vast experience with one or more of the major DBMS products, such as Oracle Database Server, Sybase Adaptive Server, Informix Dynamic Server, and Microsoft SQL Server.

The major DBMS available today are extremely complex and require a regular and demanding administration. The DBA is responsible for managing all the aspects concerning the DBMS environment, which makes the list of tasks performed by a database administrator quite extensive. Although different DBMS have different implementation and functionalities, administration tasks can be grouped in core areas common to all DBMS. The following points summarize those administration areas (each area corresponds to many administration tasks) and give an idea of the huge complexity of the administration of a database system:

- *Memory and processes administration*: The goal is to optimize I/O operations through the correct definition of the memory allocation parameters. Additionally, database administrator has to manage a number of different process types and the communications between the database clients and the server.
- *Security management*: Security is one of the most important issues in a database system administration and represents a very complex administration area that requires continuous attention from the DBA. Some typical tasks include adding and removing users, managing privileges, and monitoring resources utilization.

- *Storage administration*: The main goal is to minimize the contention when accessing the data and the fragmentation in the physical storage of the database objects. Several aspects have to be managed, such as the replication of some types of files, the distribution of the files by several disks, and other parameters related to physical space allocation.
- *Database objects administration*: The database administrator must frequently monitor and manage the database objects (tables, indexes, etc.) in order to maximize the system performance. For example, create the adequate indexes, cluster or partition some objects, and configure space allocation parameters, are some of the important tasks that DBA have to execute frequently.
- *Recovery mechanisms administration*: The correct configuration of the recovery mechanisms in order to minimize the recovery time and the lost of transactions in a failure situation is one of the most important and complex database administration aspects. However, it is also important to conciliate the aspects related to recovery with system performance. The main goal is to achieve the right balance between system recoverability and performance.

These major areas of administration can be found in any commercial DBMS, as they are related to core functions available in all DBMS [10]. The following subsection presents a comparative analysis of administration tasks for each class in three representative DBMS.

3.2. Administration in different DBMS

Table 1 presents the main administration tasks (each task can be divided in several subtasks) for three of the leading DBMS in the market: Oracle 8*i* [15], Sybase Adaptive Server 12.5 [22,23], and Informix Dynamic Server 9.3 [24]. The tasks are grouped using the administration areas presented before and have been identified based on the field experience of the authors on database administration, the comparative analysis of the administration manuals of the different products considered, and through discussions and interviews with several database administrators of real databases installations. Each column shows the administration tasks for a given DBMS and tasks in the same row are considered equivalent.

Although the details of some administration tasks are specific to each DBMS, the standard SQL used by the vast majority of DBMS greatly simplifies the establishment of equivalent tasks in different DBMS implementation. However, some of the tasks are intimately related to specific features of a given DBMS and do not have counterparts in other DBMS. There are also some tasks in a DBMS that may correspond to two or more tasks in another DBMS.

3.3. Classes of operator faults

As seen before, the list of tasks performed by a database administrator is very long. Although some of those tasks are common to several DBMS, different database systems have different possibilities for administration and consequently different sets of possible operator faults. Thus, we have decided to analyze the main database administration areas and tasks and try to map them in classes of operator faults common to all DBMS. The following points show the proposed classes of operator faults (each class of faults corresponds to many types of administrator mistakes) and gives some examples of faults for each class:

- *Memory and processes administration*: Mistakes in the administration of processes and memory structures. Deleting or corrupting the database initialization files, incorrectly define the memory allocation

Table 1
Administration tasks in different DBMS

Class	Oracle 8*i*	Sybase Adaptive Server 12.5	Informix Dynamic Server 9.3
Memory and processes administration	Install and configure the database server	Install and configure the database server	Install and configure the database server
	Create Oracle databases		
	Startup and shutdown	Startup and shutdown	Startup and shutdown
	Manage Oracle processes	Manage multiprocessor servers	Manage virtual processors and threads
	Manage SGA allocation	Configure memory	Manage shared memory
		Configure data caches	
	Manage job queues		
	Set database configuration parameters	Set database configuration parameters	Set database configuration parameters
	Manage client/server communications	Manage client/server communications	Manage client/server communications
Security management	Establish security policies	Establish security policies	Establish security policies
	Manage users and resources	Manage adaptive server logins and database users	Manage users
		Limit access to server resources	Monitor resources and user activity
	Manage user privileges and roles	Manage user permissions and roles	Grant and revoke privileges
			Create and use roles
		Manage remote servers	
	Audit database use	Audit	Audit
Storage administration		Create and manage user databases	
		Set database options	
	Manage tablespaces	Create and use segments	Manage databases
			Manage tablespaces
	Manage data files	Initialize and mirror database devices	Manage dbspaces, blobspaces, sbspaces, and extspaces
	Manage rollback segments		
	Manage temporary segments	Reorganize space in tables	Manage temporary dbspaces and temporary sbspaces
Database objects administration	Manage tables	Manage tables	Manage tables
	Manage performance of optimization objects	Manage performance of optimization objects	Manage performance of optimization objects
	Manage other objects	Manage other objects	Manage other objects
	Manage objects replication		
Recovery mechanisms administration	Develop backup and recovery strategies	Develop backup and recovery plans	Develop backup and recovery plans
	Manage control files		
	Manage the online redo log	Manage transaction log	Manage logical-log files
	Manage archived redo logs	Restore system databases	Manage the physical log

Table 1 (*Continued*)

Class	Oracle 8*i*	Sybase Adaptive Server 12.5	Informix Dynamic Server 9.3
	Perform backups and recovery	Backup and restore user databases	Perform manual recovery Manage mirroring Use high-availability data replication
	Check data block corruption	Check database consistency	Check consistency

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 management*: 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 administration*: 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 objects administration*: Errors related to the management of the user objects. The removal of a user object (e.g., table, index, cluster, etc.), and the incorrect use of the optimization structures (e.g., indexes, clusters, etc.) are common faults related to the database schema administration.
- *Recovery mechanisms administration*: 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.

3.4. Comparative analysis of DBMS administration and susceptibility to operator faults

Different DBMS include different sets of administration tasks and consequently have different sets of possible operator faults. In order to identify the differences and similarities among different DBMS concerning operator faults, a comparative analysis has been made to identify the operator faults associated to each administration task. Tables 2–4 present the list of the administration tasks and respective operator faults types identified for the three DBMS considered in this study, respectively, Oracle 8*i* [15], Sybase Adaptive Server 12.5 [22,23], and Informix Dynamic Server 9.3 [24]. Note that a given fault type actually represents many possible faults (e.g., as a typical database has hundreds of tables there are many possibilities for deleting a table by mistake).

As we can see, it is possible to establish equivalence among many operator faults in different DBMS. A very important aspect is that very sophisticated DBMS with many things to administrate seem to be more prone to operator faults than simpler DBMS, with only a few things to administrate (obviously, DBMS with less administration possibilities have limited tuning possibilities). Although a more comprehensive study is required to evaluate the portability of these types of faults across more DBMS implementations, this analysis and suggests that a faultload based on a subset of these types of operator faults could be fairly general.

The interface of the administration tools is also very important to identify weaknesses in DBMS concerning operator faults. Normally a DBMS includes several administration tools. These tools have two types of interface: graphical interface or SQL command line interface. In a graphical interface the

Table 2
Administration tasks and operator faults in Oracle 8*i* DBMS

Class	Administration tasks	Operator fault types
Memory and processes administration	Create Oracle database	Create an Oracle database during peak workload
	Startup and shutdown	Make a database shutdown inadvertently
	Manage Oracle processes	Incorrect configuration of the processes parameters
	Manage SGA allocation	Incorrect configuration of the SGA parameters
	Manage job queues	Incorrectly set a job to a peak workload time
	Set database configuration parameters	Incorrect configuration of the database parameters; remove or corrupt the initialization file
	Manage client/server communications	Incorrect configuration of the maximum number of user sessions; kill a user session
Security management	Manage users and resources	Database access level faults (passwords); delete a database user; assign incorrect profiles to users; grant incorrect disk space to users
	Manage user privileges and roles	Incorrect attribution of system privileges to users; incorrect attribution of object privileges to users; incorrect attribution of roles to users
Storage administration	Manage tablespaces	Delete a tablespace; set a tablespace offline; allow a tablespace to run out of space
	Manage data files	Delete or corrupt a data file; set a data file offline; incorrect distribution of data files through disk
	Manage rollback segments	Delete a rollback segment; set a rollback segment offline; insufficient number of rollback segments; allow a rollback segment to run out of space
	Manage temporary segments	Delete a temporary segment; allow temporary segments to run out of space
Database objects administration	Manage tables	Delete a table; incorrect configuration of table's storage parameters; setting NOLOGING option in tables
	Manage performance optimization objects	Incorrect use of performance optimization objects
	Manage other objects	Delete any database object
	Manage objects replication	Incorrectly replicate objects
Recovery mechanisms administration	Develop backup and recovery strategies	Develop a wrong backup and recovery strategy
	Manage control files	Delete a control file
	Manage the online redo log	Delete a redo log file or group; store all redo log group members in the same location; insufficient redo log groups (set of replicas of a redo log file) to support archive logs
	Manage archived redo logs	Inexistence of archive logs; delete an archive log file Store archive log files in the same disk as data files
	Perform backups and recovery	Backups missing to allow recovery; make a hot backup during peak workload
	Address data block corruption	Correct data block corruption during peak workload

Table 3
Administration tasks and operator faults in Sybase Adaptive Server 12.5 DBMS

Class	Administration tasks	Operator faults types
Memory and processes administration	Startup and shutdown	Make a database shutdown inadvertently
	Manage multiprocessors servers	Kill a process; incorrect configuration of the processes parameters
	Configure memory	Incorrect configuration of memory parameters
	Configure data caches	Incorrect configuration of data caches parameters
	Set database configuration parameters	Incorrect configuration of database parameters; remove or corrupt the configuration file
	Manage client/server communications	Kill a user session
Security management	Manage logins and database users	Database access level faults (passwords); delete a database user
	Limit access to server resources	Incorrect distribution of resources
	Manage user permissions and roles	Incorrect attribution of permissions to users; Incorrect attribution of roles to users
Storage administration	Create and manage user databases	Delete a user database
	Set database options	Incorrect configuration of database options
	Create and use segments	Delete a segment
	Initialize and mirror database devices	Delete or corrupt a database device
	Reorganize space in tables	Reorganize space in table during peak workload
Database objects administration	Manage tables	Delete a table
	Manage performance optimization objects	Incorrect use of performance optimization objects
	Manage other objects	Delete any database object
Recovery mechanisms administration	Develop backup and recovery strategies	Develop a wrong backup and recovery strategy
	Manage transaction log	
	Restore system databases	Inexistence of system databases backups
	Backup and restore user databases	Inexistence of user databases backups; make a user database backup during peak workload
	Check database consistency	Check database consistency during peak workload

database administrator executes the administration tasks by clicking on information boxes and buttons in a graphical environment. The administration tool translates those actions into SQL commands and submits them to the DBMS. In a SQL command line interface, the database administrator must write by hand the SQL commands. Obviously administration using tools with an interface based in a SQL command line is more susceptible of introduce operator faults than using graphical interfaces.

A very important aspect concerning administration tools is the existence or not of some kind of mechanisms to confirm and undo operator tasks. In tools with a confirmation mechanism, the operator has to validate each operation (through a commit command or clicking in a button), which reduces the probability of occurrence of faults. The undo mechanism is essential and allows to rollback an operation after its execution (nullifying the fault if it exists). Obviously, it is very difficult to rollback some types of complex operations, such as deleting a file, without using extremely complex recovery procedures. That is the reason why most of the DBMS do not include undo facilities as a standard administration mechanism.

Table 4
Administration tasks and operator faults in Informix Dynamic Server 9.3 DBMS

Class	Administration tasks	Operator faults types
Memory and processes administration	Startup and shutdown	Make a database shutdown inadvertently
	Manage virtual processes and threads	Kill a process; incorrect configuration of the processes parameters
	Manage shared memory	Incorrect configuration of shared memory parameters
	Set database configuration parameters	Incorrect configuration of database parameters; remove or corrupt the configuration file
	Manage client/server communications	Kill a user session
Security management	Manage users	Database access level faults (passwords); delete a user
	Monitor resources and user activity	Incorrect distribution of resources
	Grant and revoke privileges	Incorrect attribution of privileges to users
	Create and use roles	Incorrect attribution of roles to users
Storage administration	Manage databases	Delete a database
	Manage tablespaces	Delete or corrupt a tablespace
	Manage dbspaces, blobspaces, sbspaces, extspaces	Delete or corrupt a dbspace, blobspace, sbspace, or a extspace
	Manage temporary dbspaces and temporary sbspaces	Delete a temporary dbspace or a temporary sbspace
Database objects administration	Manage tables	Delete a table
	Manage performance optimization objects	Incorrect use of performance optimization objects
	Manage other objects	Delete any database object
Recovery mechanism administration	Develop backup and recovery strategies	Develop a wrong backup and recovery strategy
	Manage logical-log files	Delete a logical-log file
	Manage the physical log	Delete a physical log file
	Manage mirroring	Incorrectly mirroring of files
	Use high-availability data replication	Incorrectly replication of files
	Check consistency	Check consistency during peak workload

3.5. 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 Fig. 2 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 behavior according to the system state, which means that different instants for the injection of faults must be chosen.

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 includes 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

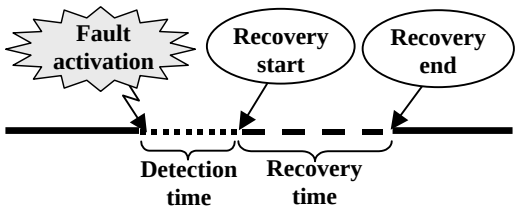

Fig. 2. Steps in the injection of an operator fault.

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 (for experiment purposes) detection time has to be established for each type of operator fault. In practice, the detection time is estimated for each fault based on previous experience and on the analysis of the possible fault effects. It is worth noting that the detection time as shown in Fig. 2 also include the time required for the DBA to understand the effects of the fault and start the adequate recovery procedure.

4. Experimental setup

The basic platform used in the experiments presented in the 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 8*i* database server is used as case study for this joint evaluation of performance and recoverability. The TPC-C benchmark in a realistic OLTP workload that represents a typical database installation. The business represented by TPC-C is a wholesale supplier having a number of warehouses and their associated sales districts, 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 found in many complex OLTP application environments. The performance metric for this benchmark is expressed in transactions-per-minute-C (tpmC).

Fig. 3 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 standby 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 re-establish the transaction execution at the client application level (i.e., as it would be seen by the end-user).

The operator faults are injected through a set of scripts as described in Section 3.5 and following the steps represented in Fig. 2. In order to make it easy to reproduce the experiments we have decided to

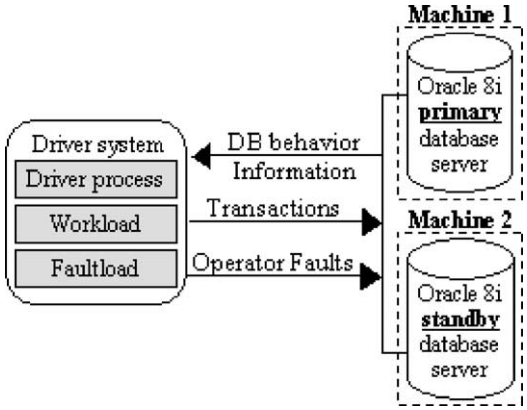

Fig. 3. Experimental setup layout.

inject the faults in three specific moments (instead of using a random distribution for the fault triggers), respectively, after 150, 300, and 600 s from the TPC-C start. The faults injected in the first instant (150 s) affect the system when it is speeding up to reach the nominal transaction performance. The faults injected in the second instant (300 s) affect the system when it is on its maximum processing throughput. Finally, the faults injected in the third instant (600 s) affect the system when a large amount of transactions has already been processed. The duration of each individual experiment was 20 min and the execution of all the experiments is fully automatic.

We have excluded from this first study the classes of faults "security management" and "recovery mechanisms administration". In fact, the security issues are out of scope of this work, while faults in the recovery mechanisms administration just affect the database performance. In fact, an erroneous change in the recovery configuration does not affect the system functionality. In this case it would be necessary a second fault of other type to activate the recovery and reveal the effects of the first fault (e.g., a incomplete recovery).

We selected six main types of fault from the three operator faults classes considered ("memory and processes administration", "storage administration", and "database object administration" faults). These fault types have been chosen based on their ability to emulate the effects of other types of faults, diversity of impact in the system, and complexity of required recovery. Although relevant, the fault types selected may present some deviation from those really experienced in the filed. A reasonable estimation of the frequency of each fault can be obtained by field data (using for example real database logs) and used in the selection of the fault types.

The basic six types of faults chosen include:

- *Shutdown abort*: This type of fault represents all kinds of operator faults (and several other faults such as loss of power, OS crash, etc.) that cause an abrupt shutdown. These faults are perceived by the users as a database crash and always require a recovery procedure to recovery the lost data.
- *Delete a data file*: This type of fault represents a frequent type of database administrator mistakes that require the use of a data file backup, followed by recovery. This type of fault also represents other common types of faults such as disk failures or file corruption due to erroneous operating system behavior.

- *Delete a tablespace*: This type of fault represents all the operator faults that affect one or more files associated to the tablespace, corruption of control files, or even faults that affect any object stored in the tablespace. This type of fault requires one of the most complex recovery procedures.
- *Set a data file offline*: This type of fault represents minor operator mistakes that require simple recovery procedures. In this case the DBMS continues working normally and only transactions that access database objects stored in the affected data file fail.
- *Set a tablespace offline*: Similar to the previous one but with more extensive effects and a more complex recovery.
- *Delete database object*: This type of fault represents all sorts of database administrator mistakes that affect the database objects. It is worth noting that a typical database include thousands of different objects (some of them requiring frequent management), which makes this type of fault one of the most frequent.

A total of 168 faults have been injected from the six types of faults mentioned above, covering all the tested configurations. Table 5 illustrates the steps required to emulate the operator faults considered. The scripts were programmed in Perl and SQL and can be easily ported to other DBMS.

Table 5
Steps required to emulate the types of operator faults considered

Fault	Steps to perform emulation	Steps to perform recovery
Shutdown abort	Shutdown abort	Startup nomount pfile = '*file_name*'
Delete a data file	Connect user/password@database Alter database data file '*file_name*' offline drop	Find tablespace corresponding to data file datafile_name Alter tablespace *tablespace_name* offline immediate Restore a backup of data file '*file_name*' Alter database recover tablespace *tablespace_name* Alter tablespace *tablespace_name* online
Delete a tablespace	Connect user/password@database Drop tablespace *tablespace_name* including contents	Shutdown Restore a backup of all data files Delete control files Create control files Alter database automatic recover database until time '*time_stamp*' Alter database open resetlogs
Set tablespace offline	Alter tablespace *tablespace_name* offline	Alter tablespace *tablespace_name* online
Set data file offline	Alter database data file '*datafile_name*' offline	Alter database recover automatic data file '*datafile_name*' Alter database data file '*datafile_name*' online
Delete database object	Drop *object_type object_name*	Shutdown Restore a backup of all data files Alter database automatic recover database until time '*time_stamp*' Alter database open resetlogs

5. Experimental results and discussion

Four sets of experiments have been conducted. The first set of experiments was used to tune the database to achieve the maximum performance, without taking into account the recovery goals. The second set of experiments evaluates the effectiveness of different configurations of the basic recovery mechanism (online redo logs) and the next set of experiments evaluates different configurations of the archive log mechanism. The last set of experiments assesses the effectiveness of the standby database mechanism. The following subsections present and discuss these results.

5.1. Results with basic recovery mechanism (online redo logs)

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 (a group is the set of replicas of a given redo log file), and the checkpoint timeout (controlled by the log_checkpoint_timeout parameter, which specifies the amount of time between checkpoints). The actual trigger of a checkpoint and the cache activity profile are also intimately related to the transaction activity (dependent, in turn, of the user applications), which is responsible for the difficulties in forecasting the impact of recovery configurations on performance (that is why we need benchmarking to assess this).

In order to assess the impact of different checkpointing policies on performance and recovery, we 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 6 shows the set of recovery configurations and Fig. 4 shows the results. Note that the performance results have been obtained without the insertion of any (artificial) faults.

Results show a clear impact of the different recovery configurations on database performance. However, only for configurations having high checkpointing rates we have observed a clear impact on the performance (configurations F1G6T1, F1G3T1, and F1G2T1). As expected, the recovery time for these configurations is the lowest. An important conclusion from Fig. 4 is that we can increase the checkpoint rate in Oracle 8*i*, reducing the recovery time, without causing a severe impact on performance (configurations from F100G3T1 to F10G3T1).

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

One very important conclusion is that all the faults represented by the shutdown abort (the ones injected in these particular experiments) 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 (i.e., this is the expected behavior when the recovery is done successfully).

Table 6
Set of recovery configurations used

Configuration	File size (MB)	Redo log groups	Checkpoint timeout (s)	#CKPT per experiment
F400G3T20	400	3	1200	1
F400G3T10	400	3	600	1
F400G3T5	400	3	300	1
F400G3T1	400	3	60	1
F100G3T20	100	3	1200	5
F100G3T10	100	3	600	5
F100G3T5	100	3	300	5
F100G3T1	100	3	60	4
F40G3T10	40	3	600	13
F40G3T5	40	3	300	12
F40G3T1	40	3	60	14
F10G3T5	10	3	300	54
F10G3T1	10	3	60	55
F1G6T1	1	6	60	319
F1G3T1	1	3	60	380
F1G2T1	1	2	60	263

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 activate the archive logs to evaluate their impact on performance and recoverability. These experiments use the configurations F40G3T10 to F1G2T1 (the other configurations are not relevant because of the large

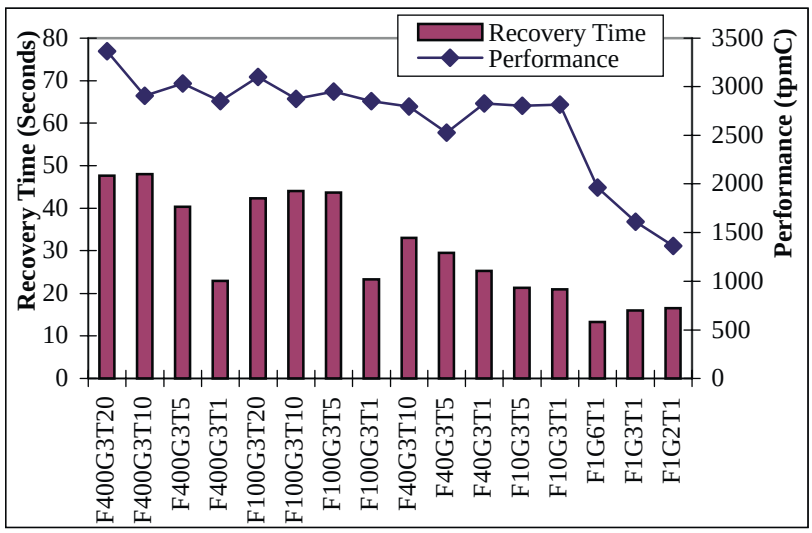

Fig. 4. Performance vs. recovery time.

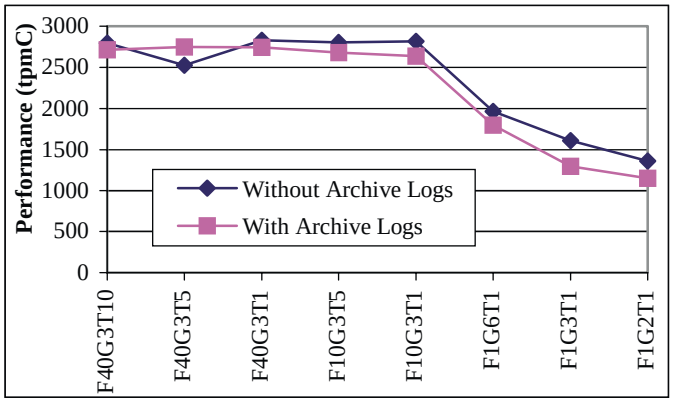

Fig. 5. Performance with and without archive logs.

size of the online redo log files would require a longer experiment time to start the log entries archiving) presented in Table 6 and include the injection of all types of operator faults presented in Section 4.

The results presented in Fig. 5, obtained without the insertion faults, show a moderate impact on database performance, which suggests that the archive log option must be always activated.

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. Fig. 6 presents the recovery times observed for the operator faults that caused incomplete recovery and Fig. 7 presents the recovery times for the faults that caused complete recovery. It is important to note that, for the configurations F1G6T1, F1G3T1, and F1G2T1, the recovery time for the faults injected in the third moment (600 s) is greater than 600 s.

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 (the types shutdown abort, set tablespace offline, and set data file offline).

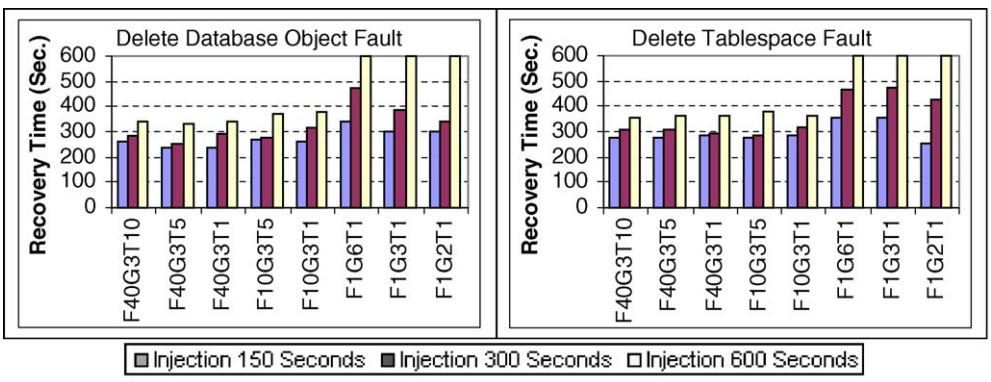

Fig. 6. Recovery time for faults with incomplete recovery.

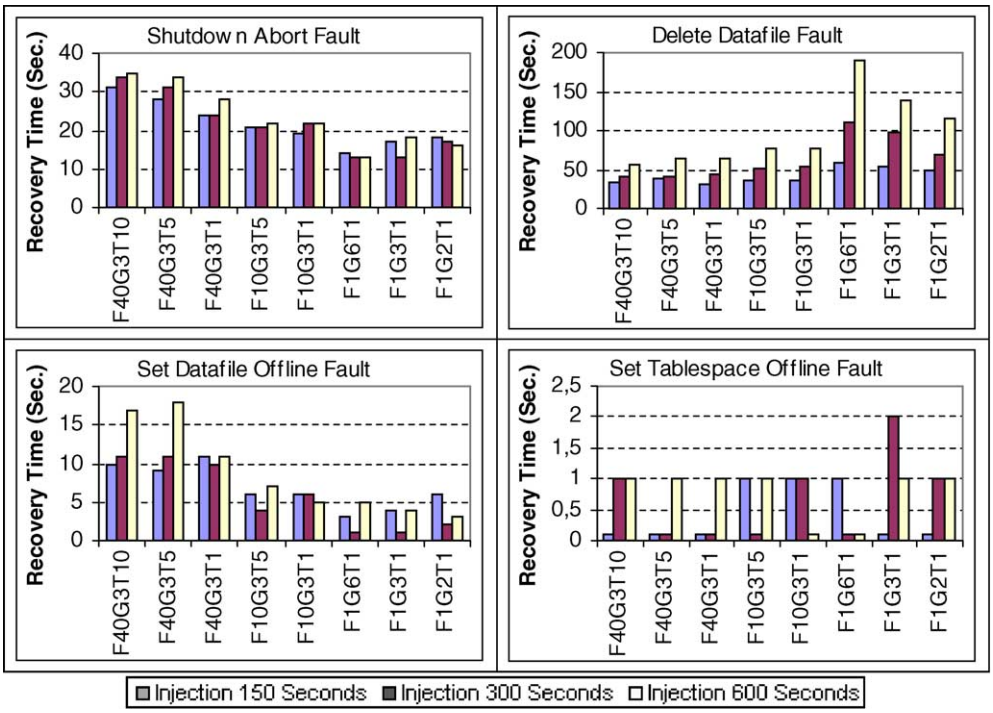

Fig. 7. Recovery time for faults with complete recovery.

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 redo log and archive log files (note that redo log and archive log files have the same size because an archive log file is always a copy of a redo log file). Small files (such as for the configurations F1G6T1 to F1G2T1) 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 archive log files (e.g., configuration F40G3T10).

A relevant result for the faults that caused incomplete recovery is that some committed transactions have been lost (i.e., could not be recovered). However, the number of lost committed transactions was constantly very small because the recovery was always started immediately after the fault occurrence. This does not 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 standby database

The main goal of the standby database is to reduce the recovery time and, consequently, minimize downtime. The standby database is kept in a permanent recovery state in which it processes the redo

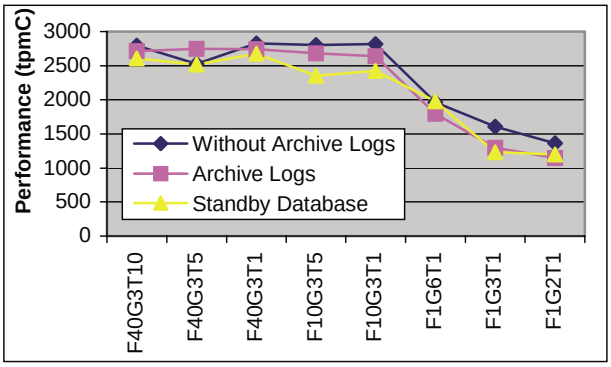

Fig. 8. Performance measured with archive logs and standby database.

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 standby database. Furthermore, in addition to the performance degradation caused by the activation of the archive logs mechanism, standby database also requires some mean to share archive log files between both machines, which also may cause some overhead. Fig. 8 shows the performance results for both the standby database and the archive log mechanisms (without faults). As we can see, both the archive logs and the standby database cause a moderate performance impact, which suggests that performance penalty is not an excuse for not using these more elaborate Oracle recovery mechanisms.

The recovery time in a standby database is equal for all the tolerated faults. This is due to the fact that the standby database activation time is independent of the primary database. Fig. 9 shows the recovery times for the operator faults injected 600 s after the workload starting. In order to allow results comparison, Fig. 9 also presents the recovery times obtained in the experiments with the archive log mechanism for the delete data file fault, also injected 600 s after the workload starting. As can be seen a considerable reduction of the recovery time is achieved.

However, in the standby database configuration, if the primary database's current redo log group cannot be archived due to the crash of the system, the transactions associated to the log entries saved on that

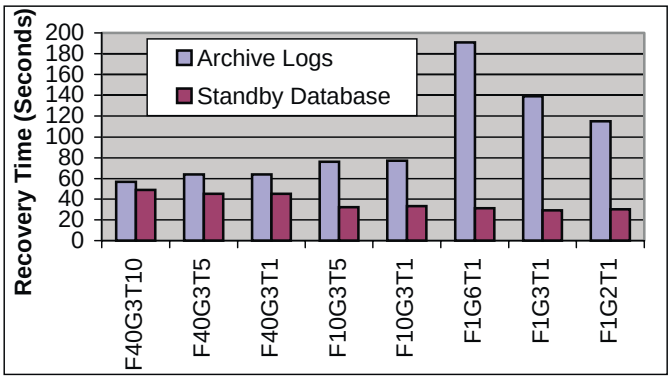

Fig. 9. Standby database recovery time vs. recovery time with archive logs.

Table 7
Possible configuration scenarios summary

Tuning goals	Configuration			Some results		
	Best configuration	Archive logs	Standby database	Performance (tpmC)	Recovery time (s)	Lost transactions
Basic recovery mechanisms (online redo logs)						
Maximize performance	F400G3T20	Do not activate	Do not activate	3367	47	None
Maximize performance and minimize recovery time at once	F1G6T1	Do not activate	Do not activate	1963	14	None
Minimize recovery time	F1G6T1	Do not activate	Do not activate	1963	14	None
	F1G3T1			1608	15	
	F1G2T1			1362	15	
Archive logs						
Maximize performance	F40G3T10	Activate	Do not activate	2713	Depends on the fault, see Tables 5 and 6	On the faults with incomplete recovery; it depends on the administrator response time
Maximize performance and minimize recovery time at once	F10G3T1	Activate	Do not activate	2637		
Minimize recovery time	F10G3T5	Activate	Do not activate	2679		
	F10G3T1			2637		
Standby database						
Maximize performance	F40G3T10	Activate	Activate	2607	49	4304
Maximize performance and minimize recovery time at once	F10G3T1	Activate	Activate	2423	33	1166
Minimize recovery time and lost transactions	F1G6T1	Activate	Activate	1979	31	173
	F1G3T1			1231	29	82
	F1G2T1			1198	30	101

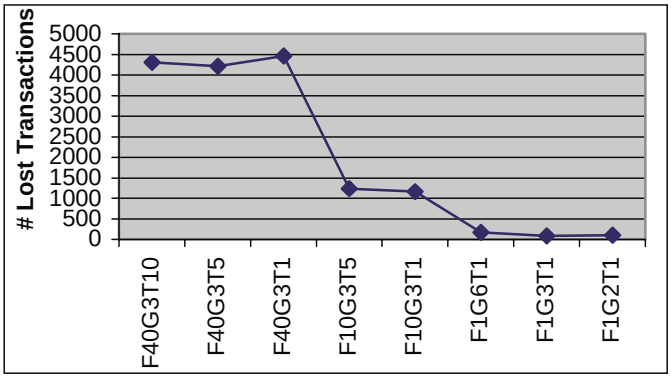

Fig. 10. Lost transactions in the standby database.

group are lost, and the corresponding committed 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.

It is worth noting that the reduction of the size of the redo log files has a clear impact on the system performance, as can be seen Fig. 8. This can be minimized by the addition of more redo log groups, but the impact on the performance is always considerable (see performance for the configurations F1G6T1 and F1G3T1).

Fig. 10 shows the results concerning the lost transactions using different redo log files sizes and different number of redo log groups. Clearly, the size of the redo log files has a dramatic effect on the reduction of the number of lost transactions.

6. Configuration scenarios summary

Table 7 summarizes the suggested configurations for the TPC-C workload considering different tuning goals (maximize performance, minimize recovery time, and the best tradeoff between these two goals) for each main configuration of recovery mechanisms presented in Sections 5.2–5.4.

7. 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 the balance between performance and recoverability need not be taken into account) 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/administrator 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.

The paper ends with a summary of suggested configurations for the TPC-C workload considering different tuning goals.

Acknowledgements

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.

References

[1] Transaction Processing Performance Consortium, TPC Benchmark C, Standard Specification, Version 5.0, 2001. http://www.tpc.org/tpcc/.
[2] W.T. Ng, P.M. Chen, Integrating reliable memory in databases, in: Proceedings of the 1997 International Conference on Very Large Databases (VLDB), August 1997, pp. 76–85.
[3] S. Chandra, P.M. Chen, How Fail–Stop are Faulty Programs? in: Proceedings of the 28th International Symposium on Fault-Tolerant Computing, Munich, Germany, 1998, pp. 240–249.
[4] M. Sabaratnam, Ø. Torbjørsen, S. Hvasshovd, Evaluating the effectiveness of fault tolerance in replicated database management systems, in: Proceedings of the 29th International Symposium on Fault-Tolerant Computing, Madison, WI, 15–18 June 1999, pp. 306–313.
[5] D. Costa, T. Rilho, H. Madeira, Joint evaluation of performance and robustness of a COTS DBMS through fault-injection, in: Proceedings of the IEEE/IFIP Dependable Systems and Networks Conference—DSN (FTCS-30 and DCCA-8), New York, USA, 25–28 June 2000, pp. 251–260.
[6] 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, in: Proceedings of the 2001 International Conference on Dependable Systems and Networks, Gotheburg, Sweden, 1–4 July 2001, pp. 225–234.
[7] A. Brown, D. Patterson, Towards availability benchmark: a case study of software RAID systems, in: Proceedings of the 2000 USENIX Annual Technical Conference, San Diego, CA, USA, 18–23 June 2000, pp. 263–276.
[8] H. Madeira, P. Koopman, Dependability benchmarking: making choices in an *n*-dimensional problem space, in: Proceedings of the First Workshop on Evaluating and Architecting System Dependability (EASY), Jointly Organized with IEEE/ACM 28th International Symposium on Computer Architecture (ISCA) and the IEEE International Conference on Dependable Systems and Networks, DSN-2001, Göteborg, Sweden, 1 July 2001.
[9] K. Kanoun, J. Arlat, D. Costa, M. Dal Cin, P. Gil, J.-C. Laprie, H. Madeira, N. Suri, DBench: dependability benchmarking, in: Supplement of the International 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, 2nd ed., McGraw-Hill, New York, 1999.
[11] E.F. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (6) (1970) 377–387.
[12] E.F. Codd, The Relational Model for Database Management, Addison-Wesley, Reading, MA, 1990.
[13] C.J. Date, H. Darwen, The SQL Standard, 3rd ed., Addison-Wesley, Reading, MA, 1993.

[14] J. Gray, A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, Palo Alto, CA, 1993.
[15] Oracle 8*i* Server Concepts Manual, Oracle Corporation, 1999.
[16] J. Gray, A census of tandem systems availability between 1985 and 1990, IEEE Trans. Reliab. 39 (4) (1990) 409–418.
[17] M. Sullivan, R. Chillarege, Software defects and their impact on systems availability—a study of field failures on operating systems, in: Proceedings of the 21st IEEE Fault Tolerant Computing Symposium, FTCS-21, June 1991, pp. 2–9.
[18] M. Sullivan, R. Chillarege, Comparison of software defects in database management systems and operating systems, in: Proceedings of the 22nd IEEE Fault Tolerant Computing Symposium, FTCS-22, July 1992, pp. 475–484.
[19] I. Lee, R.K. Iyer, Software dependability in the tandem GUARDIAN system, IEEE Trans. Softw. Eng. 21 (5) (1995) 455–467.
[20] M. Kalyanakrishnam, Z. Kalbarczyk, R. Iyer, Failure data analysis of a LAN of Windows NT based computers, in: Proceedings of the Symposium on Reliable Distributed Database Systems, SRDS18, Switzerland, October 1999, pp. 178–187.
[21] Sunbelt International, NT Reliability Survey Results, 23 March 1999. http://www.sunbelt-software.com/ntrelres3.htm.
[22] Sybase Adaptive Server 12.5, Administration Guide, vol. 1, Sybase, 2001.
[23] Sybase Adaptive Server 12.5, Administration Guide, vol. 2, Sybase, 2001.
[24] Informix Dynamic Server 9.3, Administrator's Reference, Informix, 2001.

Marco Vieira is a teaching assistant in the Department of Computer and Systems Engineering of the Polytechnic Institute of Coimbra and a Ph.D. student in the Department of Computer Engineering of the University of Coimbra, Portugal. He has been involved in the research on dependable computing since 1999 and has received an M.Sc. in computer engineering from the University of Coimbra in April 2003. His main research interests include experimental evaluation of dependable computer systems, transactional systems dependability, and dependability benchmarking.

Henrique Madeira is an assistant professor in the Department of Computer Engineering of the University of Coimbra, Portugal, where he has been involved in the research on dependable computing since 1986. He received his Ph.D. in computer engineering from the University of Coimbra in January 1994. His main research interests focus on experimental evaluation of dependable computing systems, fault injection, error detection mechanisms, and transactional systems dependability, subjects on which he has authored or co-authored more than 80 papers in refereed conferences and journals.