Fast-Start: Quick Fault Recovery in Oracle

Tirthankar Lahiri
Oracle Corporation
tirthankar.lahiri@oracle.com

Amit Ganesh
Oracle Corporation
amit.ganesh@oracle.com

Ron Weiss
Oracle Corporation
ron.weiss@oracle.com

Ashok Joshi[1]
NuGenesis Technologies Corporation
AJoshi@NuGenesis.com

ABSTRACT

Availability requirements for database systems are more stringent than ever before with the widespread use of databases as the foundation for ebusiness. This paper highlights *Fast-Start™ Fault Recovery*, an important availability feature in Oracle, designed to expedite recovery from unplanned outages. Fast-Start allows the administrator to configure a running system to impose predictable bounds on the time required for crash recovery. For instance, fast-start allows fine-grained control over the duration of the roll-forward phase of crash recovery by adaptively varying the rate of checkpointing with minimal impact on online performance. Persistent transaction locking in Oracle allows normal online processing to be resumed while the rollback phase of recovery is still in progress, and fast-start allows quick and transparent rollback of changes made by uncommitted transactions prior to a crash.

1. INTRODUCTION

The importance of availability continues to grow as more and more enterprises require continuous access to business-critical data. Availability requirements span all levels of a system, including the hardware and network, Operating System, Database System, Middleware and Application. The DBMS is a vital part of the availability stack since it protects business data from failures at all levels below the application and middleware. It is important that the frequency of failures in a DBMS be low, and that the time to recover from any failures that do occur be low so that unplanned downtime is minimized. This paper focuses on the *Fast-Start* feature of Oracle. Fast-Start is an architecture designed to provide quick recovery from the most common of unplanned outages: a system crash, without requiring any maintenance downtime during normal operation.

DBMSs typically employ a no force-on-commit policy [3] for performance reasons, i.e., buffers modified by transactions are not forced to disk when a transaction commits. Therefore, the database requires recovery following a system crash. The goal of crash recovery is to restore the physical contents of the database to a transactionally consistent state. Recovery must apply all changes made by committed transactions that were not written to disk prior to the crash (roll-forward), and erase all on-disk changes made by uncommitted transactions prior to the crash (rollback). Given that database buffer caches on modern enterprise-class servers can be many gigabytes in size and can have thousands of concurrent transactions, the time to recover following a crash can be prohibitive since millions of random disk accesses may be involved.

Fast-Start roll-forward mechanisms allow the administrator to adjust a running Oracle server to impose predictable bounds on the time required to roll-forward following a crash so that the amount of unplanned downtime can be minimized. Fast-Start rollback mechanisms also allow quick rollback of uncommitted changes while allowing online access to the database while rollback is still in progress.

The remainder of this paper is organized as follows: Section 2, provides a brief overview of the Oracle crash recovery subsystem. Section 3 describes the Fast-Start checkpointing mechanism for limiting recovery roll-forward times. Section 4 describes Fast-Start Rollback mechanisms for efficient rollback of uncommitted transactions. Section 5 provides a broad overview of Oracle's availability mechanisms. Finally, Section 6 concludes.

2. OVERVIEW OF ORACLE CRASH RECOVERY

This section contains a brief overview of the Oracle crash recovery subsystem. Further details on the architecture of Oracle can be found in [9]. Oracle supports a shared-disk architecture, and a typical database configuration consists of one or more Oracle *Instances* (a collection of user and system processes and memory) accessing a shared set of datafiles. Each instance has its own buffer cache and the different caches are kept consistent through a distributed lock management protocol. This architecture is known as the *Oracle Parallel Server* (OPS) [1, 5, 6] configuration. The following additional concepts are relevant to the understanding of crash recovery:

- **Redo thread**: Each Oracle instance has its own independent set of log files referred to as a *thread* of redo. Changes made by transactions to buffers in the cache are recorded in the redo log for the instance. For the purposes of this paper we assume

[1] This work was performed while the author was employed by Oracle corporation.

ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA

that the redo log is logically an ever-increasing sequence of bytes, indexed by RBA (Redo Byte Address). The log position at which redo application must begin is known as the *Thread Checkpoint* RBA and the RBA of the last entry in the log is known as the *Tail* of the log.

- **Rollback Segment**: Each transaction is assigned to a *Rollback Segment*, which is a special database object. Undo for changes made by a transaction is stored in the rollback segment for that transaction. When a change is made to a block in the cache, both the redo for the change as well as the redo for the change to the undo block in the rollback segment are written to the redo log atomically. This atomic logging of the redo and undo preserves atomicity of changes.

- **Consistent-Read**: Oracle uses a version-based concurrency control system known as *Consistent-Read* [2, 6]**.** Every transaction is associated with a snapshot time, and when a transaction wishes to read a buffer, it reconstructs the version of the buffer (known as a clone) that is consistent as of the transaction's snapshot time. It does so by applying any necessary undo stored in the rollback segments to the buffer. This mechanism implies that readers never need to acquire any locks. Therefore, readers do not block writers, and vice-versa.

- **Persistent Write-Locks**: In Oracle, only writers need to acquire locks. Before a transaction can modify rows within a block, it needs to acquire row-level locks for those rows. These locks are stored within the datablocks themselves and are persistent since they can be recovered by redo-application following a crash.

When an instance in an OPS cluster detects the failure of another instance, it first suspends accesses to the database. It then applies the changes recorded in the failed instance's thread, beginning from the thread checkpoint, up to the tail of the log. Note that only the logs belonging to the failed instance need to be applied. (See [7] for an excellent discussion of issues and techniques in recovery for shared-disk database systems.)

This redo-log replay is known as *roll-forward*. Following roll-forward, the database is opened for normal online processing. In the meantime the System Monitor process (SMON) processes the rollback segments and applies undo to any blocks modified by transactions that had not committed prior to the crash. This is known as the *rollback* phase, and it occurs in the background along with normal user activity. ([8] describes a restart architecture that allows the database to be reopened before roll-forward begins, after a prior *analysis* scan through the log.)

3. FAST-START CHECKPOINTING

The time required to perform roll-forward is determined by the following two factors:

- Redo log IOs: The number of redo blocks to be processed during recovery, determined by the number of blocks in the log between the thread checkpoint and the tail of the log.

- Datafile IOs: The number of distinct data blocks that need recovery. This is basically determined by the number of buffers that were modified without being written prior to the crash.

Both of these can be controlled through periodic checkpoint writes. Oracle employs one or more Database Writer (DBW) processes to write dirty buffers in the cache. (See [2] for details on the Oracle Buffer Manager). When a buffer in the cache is first-dirtied, it is linked into a Buffer Checkpoint Queue [4] in ascending order of the RBA of the first change to the buffer (we refer to this as the low RBA of the buffer). The position of the thread checkpoint is given by the low RBA of the tail buffer in the BCQ. Writing buffers in low RBA order advances the thread checkpoint. This principle is illustrated in Figure 1 below.

If buffer b_1 is written first, the thread checkpoint will advance to position c_1. Next, if buffer b_2 is written, the checkpoint will advance to position c_2. If buffer b_3 is written next, the checkpoint will advance to c_3. This mechanism implies that each buffer write is "optimal" with regard to checkpointing: Each successive write advances the thread checkpoint [4].

Periodic writes by DBW for the advancement of the thread-checkpoint is referred to as *Fast-Start Checkpointing*. While a conventional checkpoint is exhaustive, since it has to write every dirty buffer to disk, fast-start checkpointing is incremental, writing only

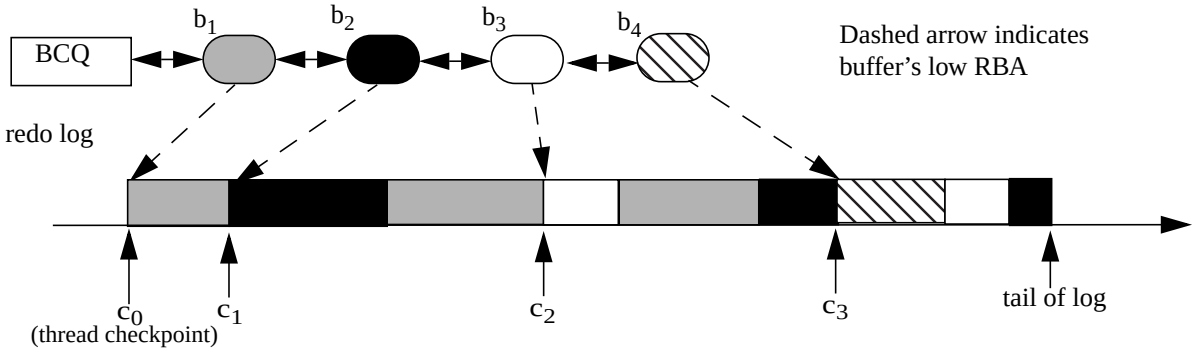

Figure 1: Buffers ordered by low RBA on Buffer Checkpoint Queue

as many buffers as is needed to advance the thread checkpoint to the necessary position in the log. Fast-Start checkpointing therefore eliminates bulk writes and the resultant I/O spikes and drop in performance that occur with conventional checkpointing.

3.1 Controlling Fast-Start Checkpointing

Instead of requiring administrators to specify a frequency for issuing (conventional) checkpoints, the fast-start mechanism provides the following dynamic configuration parameters for adaptively adjusting the rate of checkpointing to impose predictable bounds on roll-forward time:

- **FAST_START_IO_TARGET**: This parameter specifies a bound on the number of blocks that would need recovery at any given time on a running system. These blocks require random read and write IOs and dominate the roll-forward time. If an administrator believes that the I/O subsystem can perform approximately 1000 random IOs per second, she should set "FAST_START_IO_TARGET = 50000" to limit roll-forward time to approximately 50 seconds.

- **LOG_CHECKPOINT_TIMEOUT**: This parameter specifies a bound on the time elapsed since the tail of the log was at the position presently indicated by the thread checkpoint position. For example, setting "LOG_CHECKPOINT_TIMEOUT = 100" implies that the thread checkpoint should lag the tail of the log by no more than the amount of redo generated in the last 100 seconds. Since the time it takes to generate a certain amount of redo is approximately the same as the time it takes to apply that redo, LOG_CHECKPOINT_TIMEOUT may be interpreted as an upper bound on recovery time. It is an upper bound since workloads typically contain a query component that does not generate any redo.

- **LOG_CHECKPOINT_INTERVAL**: This parameter bounds the number of redo blocks between the thread checkpoint and the tail of the log. This parameter therefore imposes an upper bound on the number of redo blocks that need to be processed during roll-forward.

These parameters are dynamically adjustable while an instance is running. For instance, an administrator may set "FAST_START_IO_TARGET = 50000" in the initial configuration file when the Oracle instance is started, but later decide that 50000 IOs is too many and issue the command "ALTER SYSTEM SET FAST_START_IO_TARGET = 5000" to limit recovery IOs to 5000.

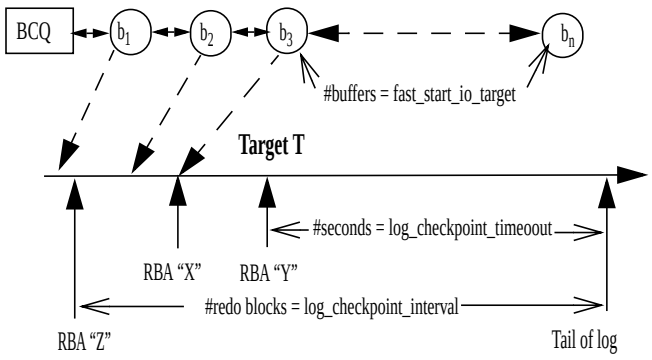

Figure 2: Target RBA calculation for fast-start checkpointing

Oracle provides the system view V$INSTANCE_RECOVERY to track the status of fast-start checkpointing activity and to allow the administrator to determine the number of log-file and data-file IOs that would be required if the instance were to crash at that specific instant. This feedback can be utilized by the administrator to dynamically adjust the fast-start checkpointing parameters to increase or decrease the rate of checkpointing activity.

3.2 Fast-Start Checkpointing Implementation

The three fast-start checkpointing parameters are used by the DBW process to periodically compute a *target RBA* in the BCQ to write buffers up to. Once DBW encounters a buffer in the BCQ with a low RBA greater than the target, it can stop writing buffers. Calculations based on each of the fast-start parameters will typically contribute a different RBA, and DBW must use the most recent RBA as its target. Each time, DBW computes the following RBAs:

- RBA X: Smallest RBA X such that number of buffers in the BCQ with RBAs less than X is equal to or less than the value of FAST_START_IO_TARGET.

- RBA Y: Smallest RBA Y such that the number of seconds elapsed since the tail of the log was at Y is equal to or less than LOG_CHECKPOINT_TIMEOUT seconds.

- RBA Z: Smallest RBA Z such that the number of redo blocks between Z and the tail of the log is equal to or less than LOG_CHECKPOINT_INTERVAL.

DBW will pick T =max(X, Y, Z) as its target RBA and write buffers on the BCQ in RBA order till it encounters a buffer whose low RBA is greater than T. This calculation is illustrated in Figure 2, in which the most aggressive target RBA is shown to be Y. This calculation is repeated by DBW each time it wakes up to write a batch of buffers, so that over time, all three fast-start roll-forward targets are met.

The I/O overhead imposed by fast-start checkpointing can be inferred from two system statistics:

- *Physical Writes*: This statistic counts all the physical writes that have occurred, both for checkpoint advancement and for aging writes for cold dirty buffers.

- *Physical Writes Non-Checkpoint*: This statistic is the theoretical number of writes that would have occurred up to this point in the absence of any checkpointing, i.e., if aging were the only reason for performing writes.

The difference between these two values is the number of extra writes that had to be performed to advance the checkpoint. A DBA can use this information to decide whether to make the fast-start target settings more or less aggressive.

3.3 Experimental Results with Fast-Start Checkpointing

Experiments have shown that running with aggressive values for the fast-start checkpointing parameters imposes minimal overheads on a running system while effectively limiting the duration of recovery roll-forward. This is because fast-start checkpointing does not add significantly to the physical I/O rate of a system.

Table 1 below summarizes data from an update-intensive OLTP workload on a single-instance system with 200,000 buffers, each 4KB in size. The workload is run with increasingly aggressive values for the FAST_START_IO_TARGET parameter. After each run, the Oracle instance is manually terminated so that crash recovery is required when the instance is restarted. For each run, we measured the throughput of the workload as well as the time required to recover from the crash following the run.

Table 1: Performance and recovery Time with Fast-Start checkpointing

FAST_START_IO_TARGET	Throughput (TPM)	Recovery Time
disabled	805.23	0:04:34
30000	804.06	0:01:10
20000	798.26	0:01:20
10000	798.00	0:00:49
1000	797.42	0:00:21

We conclude the following from the above table:

- *Fast-Start checkpointing has negligible effect on throughput*: The performance impact of the most aggressive setting of the parameter (1000) is less than 1% of the baseline performance (obtained with the fast-start mechanism disabled).

- *Fast-Start checkpointing drastically reduces recovery time*: Recovery time is reduced by more than 90% down to the sub-minute level with the most aggressive parameter value.

4. FAST-START ROLLBACK

Most database systems maintain transaction locks in memory. When locks are stored in (volatile) memory, information on whether a resource (for example, a table block or row) is locked is no longer available immediately after a system failure. These systems require that active transactions prior to a crash (which are referred to as dead transactions) must be rolled back completely before the database can be opened for new transactions.

In Oracle, since row locks are stored persistently in the same block as rows of a table or an index, the database may be opened to new transactions before the rollback phase of database recovery completes. This is known as *Deferred Rollback*, and provides significant availability benefits by allowing rollback activity to occur in the background along with normal processing.

There are two Fast-Start rollback mechanisms to reduce the duration and impact of the rollback phase:

- *Parallel rollback:* Parallelizes the rollback of uncommitted transaction across multiple concurrent recovery processes so that rollback time is minimized.

- *On-demand rollback:* Resources locked by a dead transaction are made available to new transactions before all changes made by the dead transaction are rolled back, so that the rollback phase is largely transparent to new transactions.

4.1 Fast-Start Parallel Rollback

There are two forms of parallel rollback: inter-transaction and intra-transaction parallel rollback. Inter-transaction parallel rollback partitions the transactions to be rolled back between multiple processes. With intra-transaction parallel rollback, the changes made by a single transaction are partitioned between multiple rollback processes.

- **Inter-Transaction Parallelism**: In this mode of transaction rollback, different dead transactions are rolled back using different concurrent processes. For example, consider a parallel DML operation that was running in parallel using 48 processes on an SMP machine for 1 hour. Each Oracle process would use a separate Oracle transaction to execute the DML operation. If there is a system failure after half an hour a large amount of undo may need to be applied. However, the transaction system has already partitioned the undo generated by the parallel operations into 48 independent units. The 48 units of undo work are independent because all 48 Oracle processes ran in complete isolation, i.e., they did not depend on each other in any way. This partitioning is used by the database to perform transaction rollback in parallel.

- **Intra-Transaction Parallelism:** Intra-transaction parallel rollback is performed on any single large transaction: The system automatically partitions work done by the transaction into units such that all undo operations that are dependent on each other are performed in the correct order. For example, multiple updates to the same row within a transaction are undone in the reverse of the order of the changes to the row made by the transaction.

Oracle provides another dynamic fast-start configuration parameter, FAST_START_PARALLEL_ROLLBACK, to control the number of threads that will be used to perform parallel recovery. The accepted values of this parameter are:

- FALSE: Rollback is done by one process (no parallelism), suitable for small-scale systems.

- LOW: The maximum number of parallel processes is twice the number of CPUs.

- HIGH: The maximum number of parallel processes is four times the number of CPUs.

Depending on the configured maximum number of processes, Oracle automatically allocates the necessary number of parallel recovery slaves for performing inter-transaction parallel rollback. For each transaction it also decides whether intra-transaction rollback would be more efficient and if so, it also decides on the number of parallel slaves to use for that transaction.

4.2 Fast-Start On-Demand Rollback

On-Demand Rollback improves system availability by making data available more quickly. Leaving rollback purely to background processes may result in reduced throughput for online operations. For instance, if there is a large number of dead transactions in the system, consistent read operations will become more expensive since queries will have to continuously rollback the changes of the dead transactions to restore the correct versions of data blocks they access. This extra versioning implies additional work needed to undo the changes made by dead transactions, affects response time, and pollutes the buffer cache with clone buffers created for consistent read operations. Furthermore, transactions would be unable to modify data locked by dead transactions and would be required to wait for rollback to be complete before being allowed to proceed.

Instead, on-demand rollback allows portions of dead transactions to be recovered on-demand: If a transaction that is modifying data requires a row locked by a dead transaction, it extracts undo information for the given row from the rollback segment and applies it to the row. The rest of the dead transaction is left to be recovered in the background by parallel rollback. This mechanism is similar to the mechanism of performing consistent-read. On-demand rollback therefore enables OLTP operations to immediately recover the resource that they want to acquire, mitigating the effect of deferred recovery on response time. Another benefit of this approach is that recovery time is not adversely impacted by large transactions. On-demand rollback is always enabled and requires no parameters to be set.

5. THE AVAILABILITY GESTALT

While the Oracle Fast-Start Fault Recovery architecture makes database recovery rapid and predictable, it addresses just one aspect of ensuring application and database availability. A more holistic approach to availability must be provided in any high-availability solution, since availability is defined by the end user of the system. From the user's perspective, availability is the degree to which an application or service is available with the necessary level of functionality.

The opposite of availability is downtime. The causes of downtime can be grouped into several broad categories including: system faults and crashes, data and media failures, datacenter disasters, human error, and maintenance and continuous operations. The Oracle mechanisms that address these eventualities are listed in the following sections (for more details, please refer to [9,10]).

5.1 System Faults and Crashes

System faults and crashes can be caused by a diverse set of events including temporary power outages, software crashes, and server hardware failure. As described in this paper, Fast-Start Fault Recovery makes recovery time rapid and predictable. While Fast-Start recovers the database quickly, a single system is still a single point of failure. Clustering must be used to protect against node and system failure and thus eliminate this single point of failure:

- *Cold failover clusters* can be used to detect a system outage and restart the database on a backup system.
- *Hot clusters*, where multiple systems access a shared database using Oracle Parallel Server, provide the highest level of availability and scalability.
- The *Transparent Application Failover* feature can be used with clusters to mask failures from client systems or middle-tier systems accessing the database.

5.2 Data and Media Failures

Oracle media recovery is designed to recover from any media failure and restore the database without data loss, to a transaction-consistent state. Media recovery can be done on a subset of the database and hence the database can be in production during most media recovery scenarios. Of course, it is critical that periodic backups of the database be made and included with the database server is the *Recovery Manager* software subsystem. Recovery Manager manages the entire backup, restore, and recovery process for the database and includes numerous optimizations for improved availability, recoverability, and performance.

5.3 Datacenter Disasters

Natural disasters are a fact of life that need be addressed by database technology. Oracle provides several complimentary technologies to handle these eventualities:

- *Data Guard*, which is based on a standby database approach, can be used within a single datacenter and between datacenters on different continents. The database logs are used as the medium to keep the standby database synchronized with the production database. The use of the Data Guard feature is completely transparent to the database application and imposes no performance overhead on the primary site.
- *Oracle Advanced Replication* is also used for disaster protection where a distributed database architecture best meets the needs of the enterprise.
- *Oracle Advanced Queuing* or third-party messaging technology can also be used to build message-based applications that can operate in a disconnected mode if a component of the application solution is unavailable.

5.4 Human Error

Human error represents the largest single cause of application downtime. Some of the Oracle features designed to prevent and recover from human errors are enumerated below:

- The *Database Resource Manager* prioritizes database processing according to administrator specified resource consumption plans. It therefore prevents users from consuming inappropriate amounts of service.

- *Flashback Query* allows efficient queries against earlier versions of database objects, i.e., versions consistent as of an earlier user-specified snapshot time. It therefore protects old versions of data by allowing past versions of data to be queried and reinstatiated if necessary.

- *LogMiner* provides access to logged changes made to a database and is used to audit change activity, perform performance analysis, and when necessary, to undo SQL issued against the database.

- Oracle provides a very rich set of security technology including the *Virtual Private Database* and extensive role and privileges management services.

- *Point in Time Recovery* is also available to recover the whole database or a subset to a known point in time before a problem was introduced into the database by an errant application or a user, e.g., an unintentional deletion of a critical table.

5.5 Maintenance and Continuous Operation

One of the most difficult problems in database technology is to be able to maintain database service while concurrently performing maintenance on the system. With Oracle most maintenance operations can be done online. The consistent read technology ensures that database writers do not block readers, and vice versa, so data availability is maximized during periods of bulk updates. Most parameters that control database operations are dynamic and thus can be changed without stopping or shutting down the database. CPU, memory, and storage resources can be added to the database and are quickly put to use, again without stopping or shutting down the database. One of the most powerful capabilities in this area is the ability to add and rebuild indexes, or move and reorganize tables while users are accessing and updating the objects being manipulated.

6. CONCLUSIONS

In this paper, we have described the Fast-Start mechanisms within Oracle for improving crash recovery performance and minimizing system downtime. Fast-Start checkpointing allows checkpointing to be done continuously as a background activity so that the time to roll-forward the changes in the redo log can be bounded to acceptable limits. Persistent locking in Oracle allows transaction recovery to be deferred till after the database is open, makes the database available more quickly. Fast-Start parallel rollback provides a scalable recovery mechanism that partitions the rollback activity between concurrent recovery slaves. It quickly purges the system of dead transactions, improving the overall throughput of the system. On-demand rollback allows transactions to perform partial rollback of dead transactions so that they can access resources that were locked by dead transactions. Finally, we have presented a more general overview of availability, describing how Fast-Start fits into the broader context of availability as well as highlighting other important features in Oracle that work together in providing a true high-availability solution.

REFERENCES

[1] R.Bamford, B. Klots, D. Butler, and N. Macnaughton. Architecture of Oracle Parallel Server. In *Proceedings of the Twenty-Fourth International Conference on Very Large Databases*, New York, September 1998.

[2] W. Bridge, A. Joshi, M. Keihl, T. Lahiri, J. Loaiza, and N. Macnaughton. The Oracle Universal Server Buffer Manager. In *Proceedings of the Twenty-Third International Conference on Very Large Databases*, Athens, Greece, September 1997.

[3] T. Haerder and A. Reuter. Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys*, 15(4) 287-317.

[4] A. Joshi, W. Bridge, J. Loaiza, and T.Lahiri. Checkpointing in Oracle. In *Proceedings of the Twenty-Fourth International Conference on Very Large Databases*, New York, September 1998.

[5] B. Klots, and S. Chatterjee. Cache Coherency In Oracle Parallel Server, In *Proceedings of the Twenty-Second International Conference on Very Large Databases*, Bombay, India, September 1996.

[6] T. Lahiri, A. Joshi, A. Jasuja, S. Chatterjee. 50,000 Users on an Oracle8 Universal Server Database. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, Seattle, Washington, May 1998.

[7] C. Mohan and I. Narang. Data Base Recovery in Shared Disks and Client-Server Architectures. In *Proceedings of the Twelvth International Conference on Distributed Computing Systems*, Yokohama, June 1992.

[8] C. Mohan. A Cost-Effective Method for Providing Improved Data Availability During DBMS Restart Recovery After a Failure. In *Proceedings of the Nineteenth International Conference on Very Large Databases*, Dublin, Ireland, August 1993.

[9] Oracle Corporation. *Oracle8i Concepts Release 2 (8.1.6)*. Part Number A76965-01.

[10] Oracle Corporation. *Oracle8i Backup and Recovery Guide Release 2 (8.1.6)*, Part Number A76993-01.