Miroslaw Malek
Manfred Reitenspieß
Jörg Kaiser (Eds.)

LNCS 3335

Service Availability

First International Service Availability Symposium, ISAS 2004
Munich, Germany, May 2004
Revised Selected Papers

Springer

Miroslaw Malek Manfred Reitenspieß
Jörg Kaiser (Eds.)

Service Availability

First International Service Availability Symposium, ISAS 2004
Munich, Germany, May 13-14, 2004
Revised Selected Papers

Springer

Volume Editors

Miroslaw Malek
Humboldt-Universität Berlin
Institut für Informatik Rechnerorganisation und Kommunikation
Unter den Linden 6, 10099 Berlin, Germany
E-mail: malek@informatik.hu-berlin.de

Manfred Reitenspieß
Fujitsu Siemens Computers
Munich, Germany
E-mail: manfred.reitenspiess@fujitsusimens.com

Jörg Kaiser
University of Ulm
Department of Computer Structures, Faculty of Computer Science
James-Franck-Ring, 89069 Ulm, Germany
E-mail: kaiser@informatik.uni-ulm.de

Library of Congress Control Number: 2004117793

CR Subject Classification (1998): C.2, H.4, H.3, I.2.11, D.2, H.5, K.4.4, K.6

ISSN 0302-9743
ISBN 3-540-24420-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11378730 06/3142 5 4 3 2 1 0

Program Chair's Message

The 1st International Service Availability Symposium (ISAS 2004) was the first event of its kind where a forum was provided for academic and industrial researchers and engineers who focus on next-generation solutions where services will dominate and their dependability will be expected and demanded in virtually all applications.

As with the birth of a new baby so it was with the first symposium: It was somewhat an unpredictable event and we did not really know how many paper submissions to expect. We were nicely surprised with 28 (including three invited ones), considering the rather specialized topic and short lead time to organize this meeting. We will broaden the scope of the Symposium next year by making it clear that anything that concerns computer services might be worthwhile presenting at ISAS to a good mix of academic and industrial audiences.

A significantly increased interest in dependable services should not be a surprise as we are expecting a paradigm shift where "everything" may become a service. Computer evolution began with data types and formats. Then the concept of objects was discovered and transformed later into components. A set of components (including a set of one as well) forms a service and this concept will dominate computing, ranging from sensor networks to grid computing, for the foreseeable future. In order to make services a viable replacement and/or extension to existing forms of computing they have to be highly available, reliable and secure. The main computer and communication companies, service providers, and academics are searching for innovative ways of increasing the dependability of services that are growing in complexity and will use mainly distributed resources. This trend will continue as computer services are bound to pervade all aspects of our lives and lifestyles. No matter whether we call the computing services of the future "autonomic," "trustworthy" or simply "reliable/available" the fact of the matter is that they will have to be there seven days a week, 24 hours a day, independent of the environment, location and the mode of use or the education level of the user. This is an ambitious challenge which will have to be met. Service availability cannot be compromised; it will have to be delivered. The economic impact of unreliable, incorrect services is simply unpredictable.

All submissions were subject to a rigorous review process. Hence only 15 papers were accepted. Unfortunately, many good, worthwhile manuscripts did not make it into the program due to the high quality threshold set up by the Program Committee. Each paper was reviewed by three Program Committee members. I would like to thank wholeheartedly our PC members whose hard work was exemplary. Those who spent time at the virtual PC meeting deserve an additional recognition. Our paper selection went extremely smoothly thanks to the tremendous effort of the reviewers and solid support from my secretary Sabine Becker and my Ph.D. student Nikola Milanovic of Humboldt University Berlin. Also, Prof. Joerg Kaiser from the University of Ulm deserves a special credit for editing the symposium's proceedings and preparing the Springer volume of Lecture Notes in Computer Science. I thank all of them very much. And last but not least I would like to express my gratitude to Manfred Reitenspieß whose involvement and support were very helpful throughout the program preparation process.

The attendees enjoyed the final program as well as the lively presentations, got involved in many heated discussions, struck up new frienships, and, hopefully, got inspired to contribute to next year's symposium to be held in Berlin on April 25-26, 2005.

Munich, May 13, 2004

ISAS 2004 Program Chair
Miroslaw Malek
Humboldt–Universität Berlin
Institut für Informatik
malek@informatik.hu-berlin.de

Open Specifications for Service Availability^{TM}

Manfred Reitenspieß[1], ISAS 2005 General Chair

The continuous availability of services has always been a metric for the success of telecommunications applications: the phone system must always be operational. Today, IP data network providers and enterprise IT departments face the same requirements. Service availability architectures and feature sets have traditionally been highly proprietary and customized to individual telecom equipment provider and application requirements. Each application and hardware platform had to be designed to fit with the specific service availability environment.

Today's market dynamics require companies to be able to raise the bar and meet new and aggressive time-to-market goals. By standardizing the interfaces for high-availability functions and management, the Service Availability Forum aims to create an open, off-the-shelf infrastructure for implementers and operators of highly available services.

The Service Availability Forum is unifying functionality to deliver a consistent set of interfaces, thus enabling consistency for applications developers and network architects alike. This means significantly greater reuse and a much quicker turnaround for the introduction of new products.

As the telecom and IT market recovery accelerates, meeting both functional and time-to-market goals will be essential for success. The Service Availability Forum offers a way forward for maximizing time-to-market advantage through the adoption of a consistent and standardized interface set. The set of open standard software building blocks includes functions for managing the hardware platform components (Hardware Platform Interface), high-availability service functions used by applications (Application Interface Specification), and functions for their associated management (System Management Services).

The International Service Availability Symposium 2004 brought together scientists, technical experts and strategists to discuss availability under a number of aspects:

1. Availability in the Internet and databases
2. High availability based on Service Availability Forum specifications
3. Measurements, management and methodologies
4. Networks of dependable systems
5. Standards and solutions for high availability

The Service Availability Forum is a consortium of industry-leading communications and computing companies working together to develop and publish high availability and management software interface specifications. The Service Availability Forum then promotes and facilitates specification adoption by the industry.

[1] President Service Availability Forum, Fujitsu Siemens Computers, Munich, Germany; manfred.reitenspiess@fujitsu-siemens.com

Table of Contents

Architecture of Highly Available Databases

Sam Drake[1], Wei Hu[2], Dale M. McInnis[3], Martin Sköld[4], Alok Srivastava[2], Lars Thalmann[4], Matti Tikkanen[5], Øystein Torbjørnsen[6], and Antoni Wolski[7]

[1] TimesTen, Inc, 800 W. El Camino Real, Mountain View, CA 94040, USA
drake@timesten.com
[2] Oracle Corporation, 400 Oracle Parkway, Redwood Shores, CA 94065, USA
{wei.hu, alok.srivastava}@oracle.com
[3] IBM Canada Ltd., 8200 Warden Ave. C4/487, Markham ON, Canada L6G 1C7
dmcinnis@ca.ibm.com
[4] MySQL AB, Bangårdsgatan 8, S-753 20 Uppsala, Sweden
{mskold,lars}@mysql.com
[5] Nokia Corporation, P.O.Box 407, FIN-00045 Nokia Group, Finland
matti.j.tikkanen@nokia.com
[6] Sun Microsystems, Haakon VII gt 7B, 7485 Trondheim, Norway
oystein.torbjornsen@sun.com
[7] Solid Information Technology, Merimiehenkatu 36D, FIN-00150 Helsinki, Finland
antoni.wolski@solidtech.com

Abstract. This paper describes the architectures that can be used to build highly available database management systems. We describe these architectures along two dimensions – process redundancy and data redundancy. Process redundancy refers to the management of redundant processes that can take over in case of a process or node failure. Data redundancy refers to the maintenance of multiple copies of the underlying data. We believe that the process and data redundancy models can be used to characterize most, if not all, highly available database management systems.

1 Introduction

Over the last twenty years databases have proliferated in the world of general data processing because of benefits due to reduced application developments costs, prolonged system life time and preserving of data resources, all of which translate to cost-saving in system development and maintenance. What makes databases pervasive is a database management system (DBMS) offering a high-level data access interface that hides intricacies of access methods, concurrency control, query optimization and recovery, from application developers. During the last ten years generalized database systems have also been making inroads into industrial and embedded systems, including telecommunications systems, because of the significant cost-savings that can be realized.

As databases are deployed in these newer environments, their availability has to meet the levels attained by other components of a system. For example, if a total system has to meet the 'five nines' availability requirements (99.999%), any single component has to meet still more demanding requirements. It is not unusual to require that the database system alone can meet the 'six nines' (99.9999%) availability requirement. This level of availability leaves only 32 seconds of allowed downtime over a span of a year. It is easy to understand that under such stringent requirements, all failure-masking activities (switchover, restart etc.) have to last at most single seconds rather than minutes. Such databases are called Highly Available (HA) Databases and the systems to facilitate them are called highly available database management systems (HA-DBMS).

An HA-DBMS operates in a way similar to HA applications: high availability is achieved by *process redundancy*—several process instances are running at the same time, typically, in a hardware environment of a multi-node cluster. In addition to one or more *active processes* (Actives) running a service, there are *standby processes*, or redundant active processes, running at other computer nodes, ready to take over operation (and continue the service), should the active process or other encompassing part fail (Standbys). Database processes involve data whose state and availability is crucial to successful service continuation. Therefore we talk about *data redundancy*, too, having the goal of making data available in the presence of failures of components holding the data. Models of process and data redundancy applied in highly available databases are discussed in this paper.

Product and company names that are used in this paper may be registered trademarks of the respective owners.

2 HA-DBMS for Building Highly Available Applications

In addition to the database service itself, a highly available database brings another advantage to the HA framework environment. Just as a traditional database system frees developers from mundane programming of data storage and access, an HA-DBMS frees the developers of HA applications from some low level HA programming. To illustrate this, let us have a look at two situations. In Fig. 1, an application is running in an HA framework such as the SA Forum's Availability Management Framework (AMF) [1].

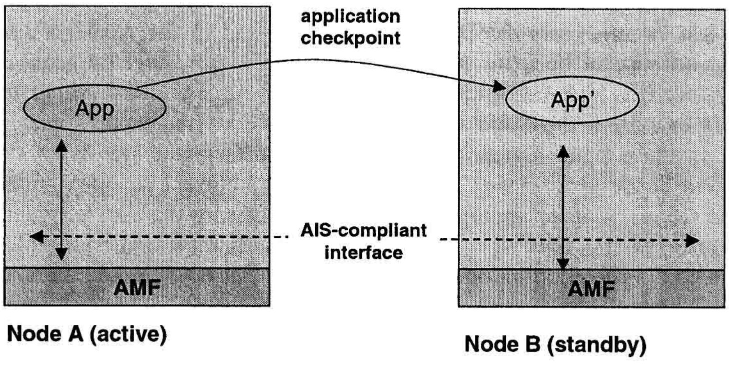

Fig. 1. An application running within AMF

Assume that the application is run in an active and a standby component (process). The application components are SA-aware meaning that they are connected to AMF in a way following the SA Forum Application Interface Specification (AIS) [1].

One demanding aspect of HA programming is to make sure that the application state is maintained over failovers. To guarantee this, application checkpointing has to be programmed into the application. The SA Forum AIS, for example, offers a checkpoint service for this purpose. Decisions have to be made about what to checkpoint and when. Also the code for reading checkpoints and recovering the application states after a failover has to be produced.

Another situation is shown in Fig. 2. In this case, the application uses the local database to store the application state, by using regular database interfaces.

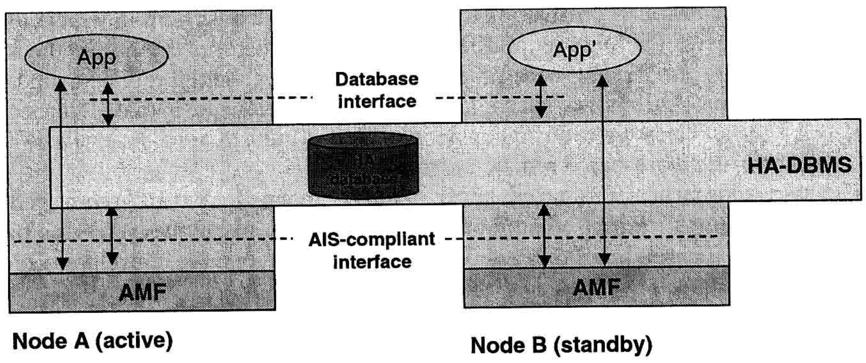

Fig. 2. A database application running within AMF

Because we deal with an HA-DBMS here, the latest consistent database state is always available after the failover at the surviving node. It is the database that does all the application checkpointing in this case. All this happens in real time and transparently. Additionally, as database systems operate in a transactional way preserving atomicity and consistency of elementary units of work (transactions), the database preserves transactional consistency over failovers, too. This way, an HA application programmer is freed from complex checkpoint and recovery programming. By bringing another level of abstraction into high-availability systems, HA-DBMS makes it easier to build highly available applications.

It should be noted, however, that the situation pictured in Fig. 2 is not always attainable. The application may have hard real-time (absolute deadlines) or soft real-time latency requirements that cannot be met by the database. Failover time of the database may be a limiting factor, too, if failover times below 100 ms are required. Finally, the program state to be preserved may not yield to database storage model. Nevertheless, the more application data is stored in a database, the more redundancy transparency is achieved.

3 HA Database Redundancy Models

Highly available database systems employ a number of redundancy concepts. All HA-DBMSs rely on having redundant database processes. When a database process dies (e.g., due to node failure), another database process can take over service. To provide correctness, each redundant process must see the same set of updates to the database. There are basically two means of ensuring this: one technique, replication, relies on the database processes to explicitly transfer updates among each other. Depending on the implementation, each replica can store its copy of the data either in main-memory or on disk. Replication is not exclusively done between individual databases. In distributed databases, one database is managed by several database processes on different nodes, with possible intra-database replication between them.

An alternate means for ensuring that all the redundant database processes see the same set of updates to the database is to rely on a shared disk system in which all the processes can access the same set of disks. Since all the processes can access the same set of disks, the database processes do not need to explicitly replicate updates. Instead, all the processes always have a single, coherent view of the data. Note that a shared disk system also has redundancy. However, it is built-in at lower levels — e.g., via RAID or by network-based remote mirroring.

The two approaches introduced above may be mapped to two known general DBMS architectures: *shared-nothing* and *shared-disk* [8], respectively. In this paper we take a more focused point of view on DBMS architectures: we concentrate exclusively on means to achieve high availability.

Several redundancy models are possible in an HA-DBMS and these are defined below. We distinguish between *process redundancy* which defines availability of the database processes and *data redundancy* which specifies, for replication-based solutions, the number of copies of the data that are explicitly maintained. Both process redundancy and data redundancy are necessary to provide a HA Database Service.

3.1 Process Redundancy

Process redundancy in an HA-DBMS allows the DBMS to continue operation in the presence of process failures. As we'll see later, most process redundancy models can be implemented by both shared-disk and replication-based technologies.

A process which is in the *active* state is currently providing (or is capable of providing) database service. A process which is in the *standby* state is not currently providing service but prepared to take over the active state in a rapid manner, if the current active service unit becomes faulty. This is called a *failover*. In some cases, a new type of process, a *spare process* (or, Spare) may be used. A spare process may be implemented as either a running component which has not been assigned any workload or as a component which has been defined but which has not been instantiated. A spare may be elevated to Active or Standby after proper initialization.

Process redundancy brings the question of how (or if) redundancy transparency is maintained in the HA-DBMS. Of all running processes, some may be active (i.e. providing full service) and some not. In the case of failovers active processes may change. The task of finding relevant active processes may either be the responsibility of applications, or a dedicated software layer may take care of redundancy transparency.

3.2 Data Redundancy

Data redundancy is also required for high availability. Otherwise, the loss of a single copy of the data would render the database unavailable. Data redundancy can be provided at either the physical or the logical level.

3.3 Physical Data Redundancy

Physical data redundancy refers to relying on software and/or hardware below the database to maintain multiple physical copies of the data. From the perspective of the database, there appears to be a single copy of the data. Some examples of physical data redundancy include: disk mirroring, RAID, remote disk mirroring, and replicated file systems.

All these technologies share the common attribute that they maintain a separate physical copy of the data at a possibly different geography. When the primary copy of the data is lost, the database processes use another copy of the data. These technologies can differ in terms of the failure transparency that is supported. Disk mirroring and RAID, for example, make physical disk failures completely transparent to the database.

Physical data redundancy is frequently combined with process redundancy by using a storage area network. This allows multiple nodes to access the same physical database. If one database server fails (due to a software fault or a hardware fault), the database is still accessible from the other nodes. These other nodes can then continue service.

3.4 Logical Data Redundancy Using Replication

Logical data redundancy refers to the situation where the database explicitly maintains multiple copies of the data. Transactions applied to a primary database D are replicated to a secondary database D' which is more or less up-to-date depending on the synchrony of the replication protocol in the HA Database. In addition to inter-database replication, intra-database replication is used in distributed database systems to achieve high availability using just one database. Note that we speak about replication in general terms since the replication scheme is vendor specific (based on the assumption that both database servers are from the same vendor). The replication can be synchronous or asynchronous, be based on forwarding logs or direct replication as part of the transaction, transactions can be batched and possibly combined with group commits. The method chosen depends on the database product and the required level of *safeness* [2]. With a *1-safe* replication ("asynchronous replication") transactions are replicated after they have been committed on the primary. With a *2-safe* replication ("synchronous replication") the transactions are replicated to the secondary, but not yet committed, before acknowledging commit on the primary. With a *2-safe committed* replication transactions are replicated and committed to the secondary before acknowledging commit on the primary. In the *very safe* replication all operations but reads are disabled if either the primary or the secondary becomes unavailable. An overview 1-safe and 2-safe methods is given in [14]. Various optimizations are proposed in [5],[4], [10] and [21]. Although most of the work on safeness-providing methods has been done in the context of remote backup, the results are applicable to in-cluster operation too.

4 Data Redundancy Models

For the rest of this paper, data redundancy refers to *logical* data redundancy. It represents the number of distinct copies of data that are maintained by the database processes themselves via replication. It does not count the copies that may be maintained by any underlying physical data redundancy models. For example, two copies of the data that is maintained by a disk array or by a host-based volume manager would be counted as one copy for the sake of this discussion, while two copies of the data maintained by the database would count as two. Note that in both cases, the loss of one copy of the data can be handled transparently without loss of availability.

We discuss data redundancy models in detail first because this is an area that is fairly unique to HA-DBMSes.

4.1 Database Fragments, Partitioning, and Replication of Fragments

To define the data redundancy models we need to define what we are actually replicating, i.e. *database fragments*[1]. Database *fragmentation* is a decomposition of a database D into fragments $P_1...P_n$ that must fulfill the following requirements:

1. *Completeness.* Any data existing in the database must be found in some fragment.
2. *Reconstruction.* It should be possible to reconstruct the complete database from the fragments.
3. *Disjointness.* Any data found in one fragment must not exist in any other fragment[2].

The granularity of a fragment is typically expressed in terms of the data model used. In relational databases, fragments may be associated with complete SQL schemas (called also catalogs) or sets of tables thereof. The lowest granularity achieved is usually called *horizontal* or *vertical* fragmentation where "horizontal" refers to dividing tables by rows and "vertical"—by columns. Note that this definition of fragmentation does not exclude viewing the database as one entity if this is a required logical view of the database.

A non-replicated, *partitioned database* contains fragments that are allocated to database processes, normally on different cluster nodes, with only one copy of any fragment on the cluster. Such a scheme does not have strong HA capabilities. To achieve high availability of data, *replication of database fragments* is used to allow storage and access of data in more than one node. In a *fully replicated* database the database exists in its entirety in each database process. In a *partially replicated* database the database fragments are distributed to database processes in such a way that copies of a fragment, hereafter called *replicas*, may reside in multiple database processes.

In data replication, fragments can be classified as being *primary* replicas (Primaries) or *secondary* replicas (Secondaries). The primary replicas represent the actual

[1] Fragment is a generalization of the common definition of table fragmentation in relational databases.

[2] This normally applies to horizontal fragmentation, but it does not exclude vertical fragmentation if we consider the replicated primary key to be an identifier of data instead of data itself.

data fragment[3] and can be read as well as updated. The secondary replicas are at most read-only and are more or less up to date with the primary replica. Secondary replicas can be promoted to primary replicas during a *failover* (see section 0).

4.2 Cardinality Relationships Among Primaries and Secondaries

1*Primary/1*Secondary
Here every fragment has exactly one primary replica which is replicated to exactly one secondary replica. This is a very common redundancy model since two replicas has been found adequate for achieving high-availability in most cases.

1*Primary/Y*Secondary
Here every fragment has exactly one primary replica and is replicated to a number of secondary replicas. This model provides higher availability than 1*Primary/ 1*Secondary and allow for higher read accessibility if secondary replicas are allowed to be read.

1*Primary
Here every fragment exists in exactly one primary replica. This model does not provide any redundancy at the database level. Redundancy is provided below the database by the underlying storage. It is used in shared disk systems and also in centralized or partitioned databases.

X*Primary
Here every fragment has a number of primary replicas and is used in N*Active process redundancy models (sometimes called multi-master). This model allow for higher read and update accessibility than 1*Primary if the same fragment is not attempted to be updated in parallel (since this would lead to update conflicts).

4.3 Relationships Between Databases and Fragments

Non-partitioned Replicated Database
The most common case is when the database and the fragment are the same. Consequently, the whole database is replicated to the Secondary location (Fig. 3). NOTE: all cases in this subsection are illustrated assuming the 1*Primary/1*Secondary cardinality.

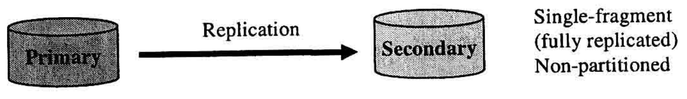

Fig. 3. Non-partitioned database

Partitioned Replicated Database
In this model, there are fragments having the purpose of being allocated to different nodes or of being replicated to different nodes (Fig. 4).

[3] If a primary replica is not available then the fragment is not available, thus the database is not available.

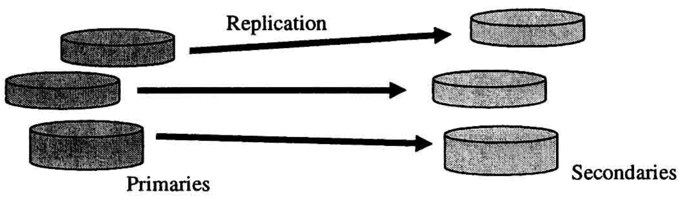

Fig. 4. Partitioned database

Mixed Replicated Fragments

A special case of a partitioned database is a database with mixed partitions whereby a database may host both Primaries and Secondaries. A special case is two databases with symmetric fragments (Fig. 5).

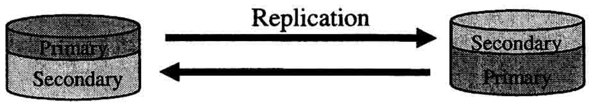

Fig. 5. Two databases with symmetric fragments

5 Process Redundancy Models

5.1 Active/Standby (2N)

Active/Standby (sometimes referred to as 2N) is a process redundancy model for HA-DBMS that is supported by both replication and shared-disk systems. Each active database process is backed up by a standby database process on another node. In Fig. 6, a replication-based example is shown while Fig. 7 provides a shared-disk based example. All updates must occur on the active database process; they will be propagated via replication, or via a shared disk, to the standby database process.

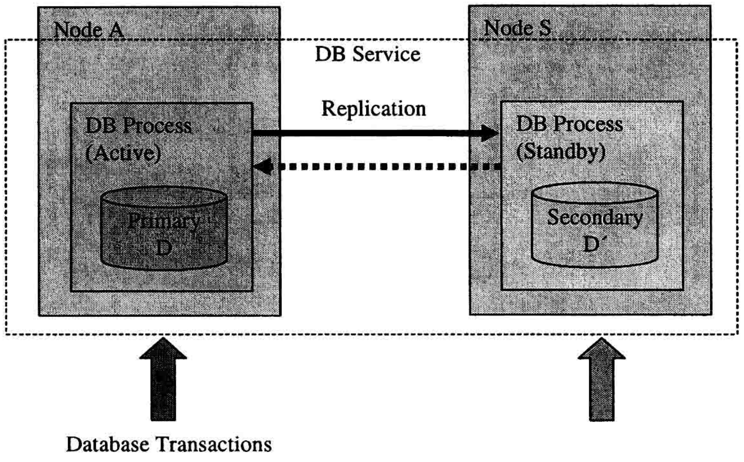

Fig. 6. Active/Standby Redundancy Model using Replication

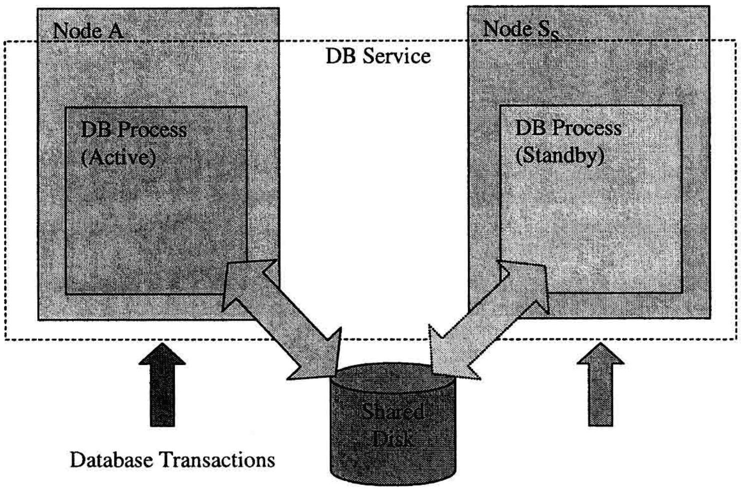

Fig. 7. Active/Standby Redundancy Model using Shared Disk

In the case of a failure of the active database process (for any reason such as software fault in the database server or hardware fault in the hosting node) the standby database process will take over and become the new active database process (Fig. 8). If the failed database process recovers it will now become the new standby database process and the database processes have completely switched roles (Fig. 9). If the HA Database has a *preferred active* database process it can later switch back to the original configuration.

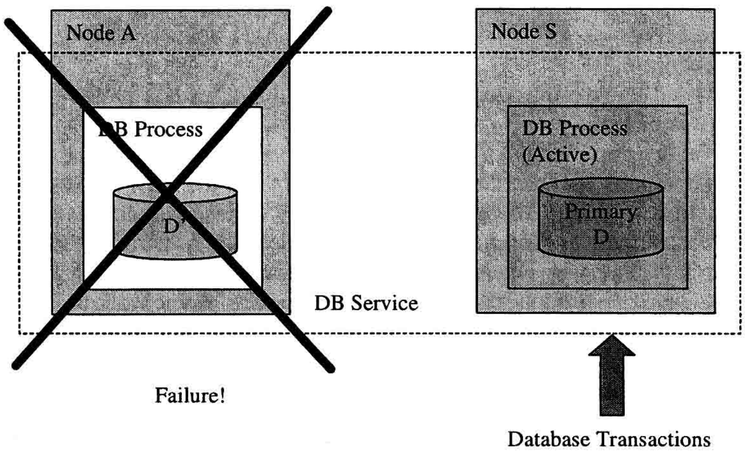

Fig. 8. Failure of Active Primary, Switchover

The standby database process can be defined as more or less ready to take over depending on the chosen safeness level and the HA requirements of the applications. To classify the non-active database processes we separate between *hot standby* and *warm standby*.