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

M. Malek et al. (Eds.): ISAS 2004, LNCS 3335, pp. 1 – 16, 2005.
© Springer-Verlag Berlin Heidelberg 2005

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].

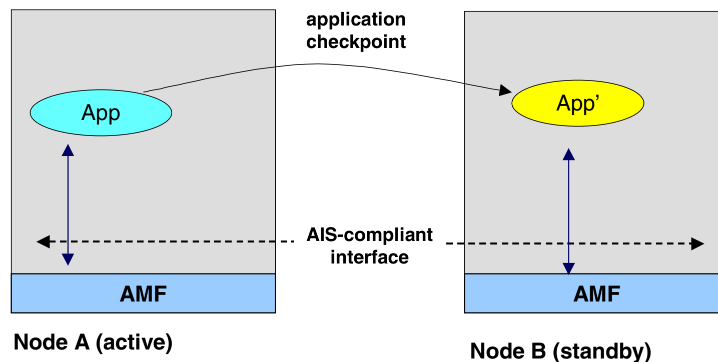

Fig. 1. An application running within AMF

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.

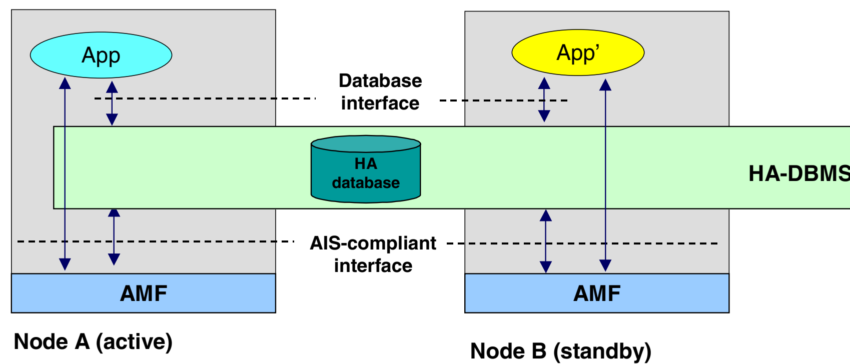

Fig. 2. A database application running within AMF

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 $P_1 \ldots P_n$ 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.

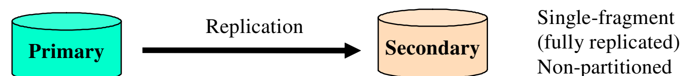

Fig. 3. Non-partitioned database

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.

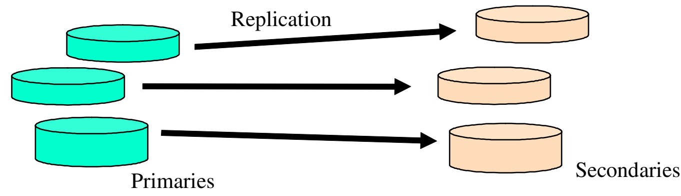

Fig. 4. Partitioned database

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).

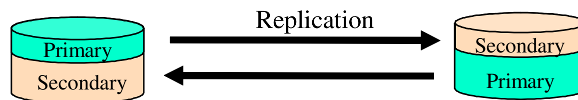

Fig. 5. Two databases with symmetric fragments

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.

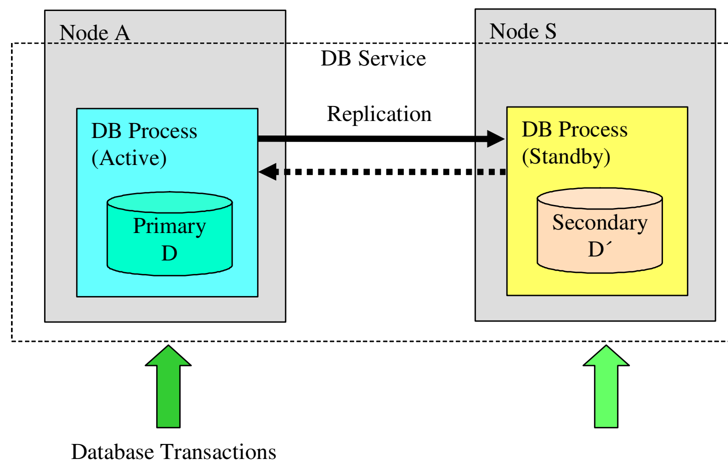

Fig. 6. Active/Standby Redundancy Model using Replication

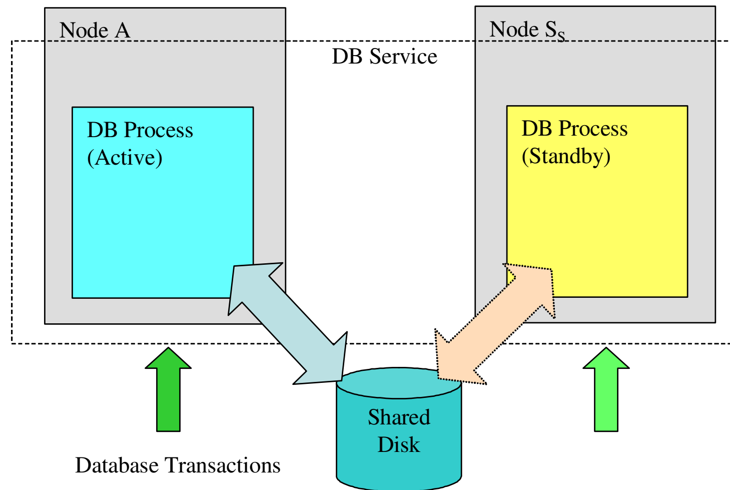

Fig. 7. Active/Standby Redundancy Model using Shared Disk

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.

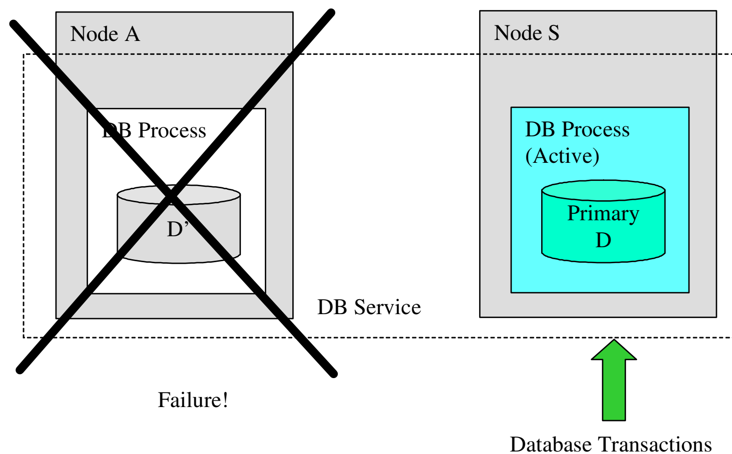

Fig. 8. Failure of Active Primary, Switchover

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*.

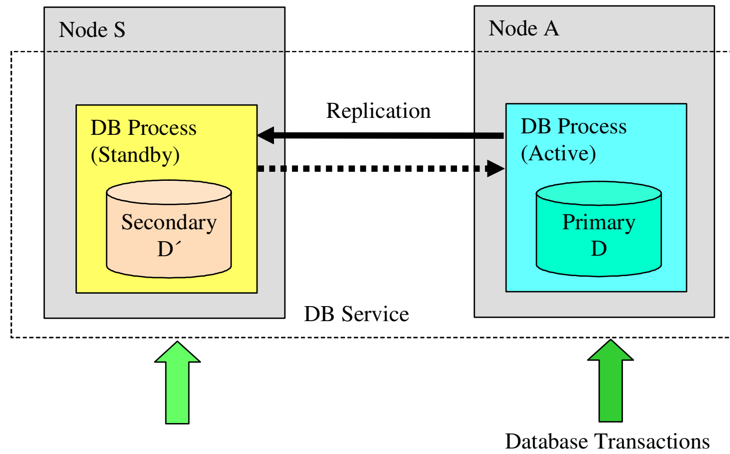

Fig. 9. Reversed Roles

Hot Standby

One active database process is being backed up by a standby database process that is ready to more or less instantly (in sub-second time) take over in case the active database process fails. The applications can already be connected to the standby or be reconnected to the standby (now active).

Warm Standby

One active database process is being backed up by a standby database process that is ready to take over after some synchronization/reconnect with applications in case the active database process fails. In this case, the failover may last from few tens of seconds to few minutes.

In the next section we introduce spares and we distinguish between standbys and spares since it is possible to have a model Active/Standby/Spare.

There are many commercial incarnations of active/standby HA database systems. In Oracle Data Guard [12] and MySQL replication [11], the active primary database ships transactions to one or more standby databases. These standby databases apply the transactions to their own copies of the data. Should the primary database fail, one of these standby databases can be activated to become the new primary database. Oracle Data Guard also supports both synchronous and asynchronous log shipping along with use selectable safeness level ranging from 1-safe to very-safe. The Carrier Grade Option of the Solid Database Engine [16] also uses an active-standby pair with a fully replicated database and dynamically controlled safeness level.

5.2 Active/S*Spare

Active/S*Spare (one Active and S Spares) is a configuration in which several *spare* database processes are pre-configured on some node(s). It is supported both by shared disk systems and replicating systems. An example of a shared-disk based architecture with a spare process is shown below (Fig. 10).

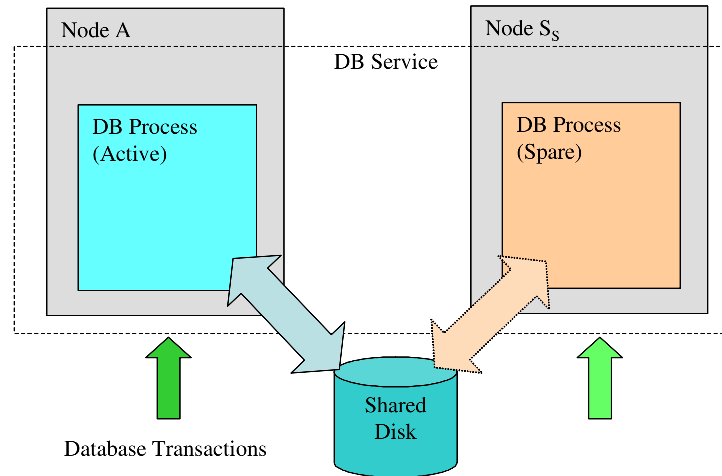

Fig. 10. Active/S*Spare Redundancy Model using Shared Disk

In a shared-disk database, if the active database process fails, the shared disk containing the database files is mounted on the node hosting the Spare (a spare node), theSpare becomes initialized with the database, and the database becomes active on that node. If the failed node restarts it will now become a new spare node. The nodes have therefore completely switched roles. If the HA Database has a preferred active database process it can later switch back to the original configuration.

In a replicating HA-DBMS, the Spare gets the database before becoming Active. The level of availability offered by this model is lower that that of Active/Standby because of the additional time needed to initialize the Spare.

This kind of operation represents the model that is supported by commercially available clustering frameworks such as Sun Cluster [17] and IBM HACMP [7]. These clustering frameworks support all the major DBMSes.

5.3 N*Active

In larger clusters, the database system can utilize more than two nodes to better use the available processing power, memory, local disks, and other resources of the nodes, to balance the load in a best possible way. In the N*Active (sometimes referred to as N-Way Active) process redundancy model, N database processes are active and applications can run transactions on either process. Here all processes support each other in case of a failure and each can more or less instantly take over. All committed changes to one database process are available to the others and vice versa. In shared-disk systems, all the database processes see the same set of changes. In a replication-based system, all changes are replicated to all the processes.

The database is fully available through all database processes and all records can be updated in all processes. In case of simultaneous conflicting updates, copy consistency may be endangered, in a replicating system. This is taken care of with a distributed concurrency control system (e.g. lock manager) or a copy update reconciliation method. In a shared disk system, the database infrastructure may be simpler because the data

objects are not replicated. The database internally implements a lock manager to prevent conflicting updates to the same data. Fig.11 shows a shared disk based N*Active model. Note that we used 2 nodes as an example even though the model supports more than 2 nodes. Fig. 12 shows a replication-based 2-node N*Active model.

There are several commercial implementations of N*Active HA database systems. The Oracle Real Application Clusters [13] is an example of an N*Active configuration whereby all the instances of the database service are active against a single logical copy of the database. This database is typically maintained in a storage-area-network (SAN) attached storage. The HADB of Sun ONE [18][6] uses also the N*Active approach that can be applied to the whole database of fragments thereof. MySQL Cluster [3][19] has an N*Active configuration in which the processes are partitioned into groups. Each operation of a transaction is synchronously replicated and committed on all processes of a group before the transaction is committed. MySQL Cluster provides a 2-safe committed replication if the group size is set to two.

N*Active configurations have demonstrated scalability with real applications. SAP, for example, has certified a series of Oracle Real Application Clusters-based SAP SD Parallel Standard Application benchmark that demonstrates near linear scalability from 1 through 4 nodes [15]. In the TPCC benchmark, a 16-node Oracle Real Application Clusters demonstrated 118% of the throughput of a single multiprocessor that contains the same number of CPUs[20].

If the database is not fully replicated and there are mixed fragments in all databases, the process model is always N*Active. For example, in Fig. 13, a 2*Active HA-DBMS is shown utilizing symmetric replication. With symmetric replication, concurrency control problems are avoided and the advantage of load balancing is retained.

Unlike most N*Active environments, in Fig. 13, the partitioning scheme is visible to the application and is often based on partitioning the primary key ranges. The applications are responsible for accessing the correct active process. Inter-partition transactions are normally not supported.

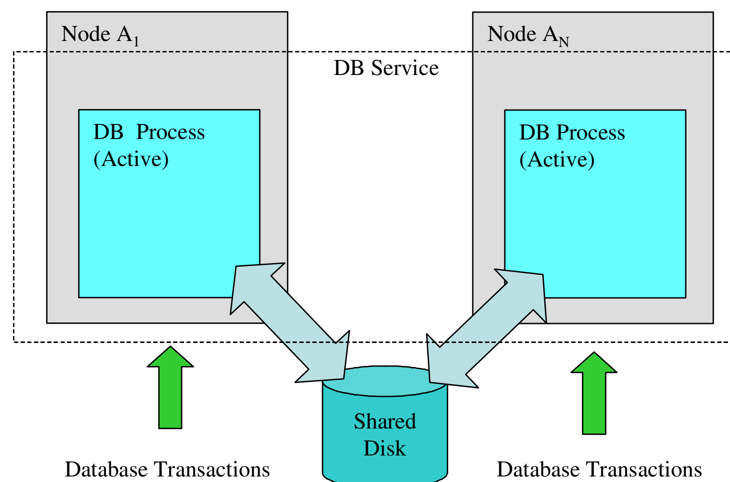

Fig. 11. N*Active, Shared Disk

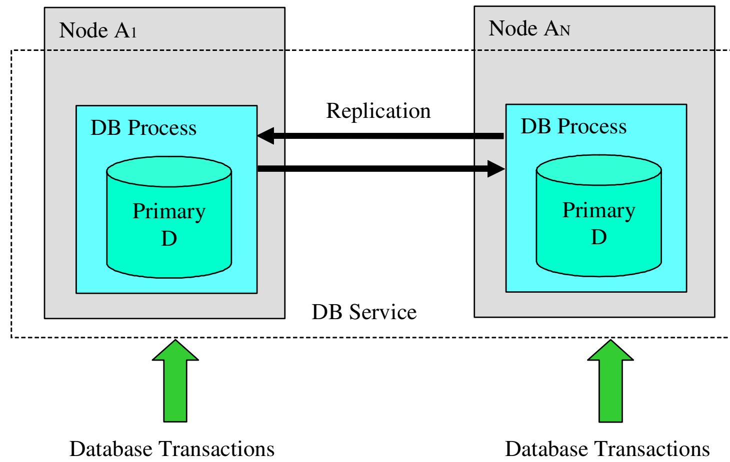

Fig. 12. N*Active, Full Replication, Redundancy Model

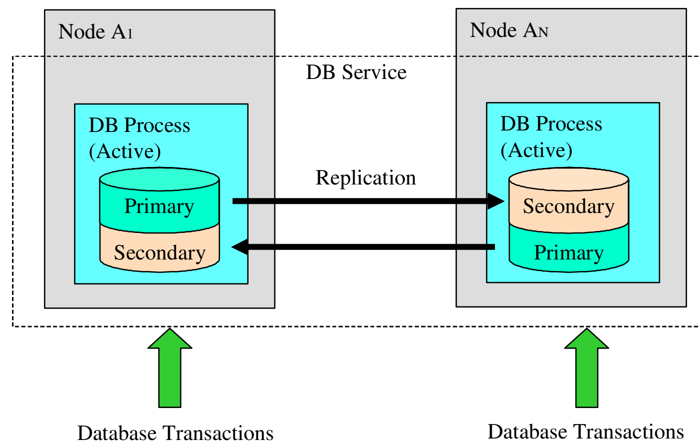

Fig. 13. A 2*Active symmetric replication database system

5.4 N*Active/S*Spare

N*Active/S*Spare (N times Active and S times Spare) is a variant of Active/S*Spare where N Active database processes provide availability to a partitioned database. As in the N*Active model, the active processes may rely on a shared disk, may use fully replicated databases or mixed fragments (partially replicated databases).

An example of a partially (symmetrically) replicated database with Spares is shown in Fig.14.

Each database process maintains some fragments of the database and Spare processes can take over in case of failure of active database processes. A Spare must get the relevant fragments of the active database process at startup.

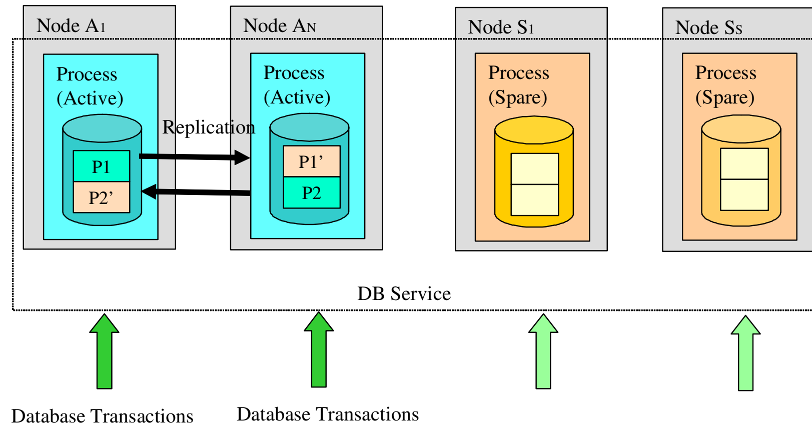

Fig. 14. N*Active/S*Spare Redundancy Model, Partially Replicated Database

5.5 Other Redundancy Models

Some systems combine multiple redundancy models to achieve different degrees of data and process redundancy. M-standby, cascading standby and geographically replicated N*active clusters [9] are several examples. Since they are composed of the other redundancy models presented in this paper, they will not be further discussed.

6 Application View

Applications may or may not be aware of the redundancy models used by various components of the database system. In Active/Standby configurations, applications normally need to be aware so that they can connect to the active instance. Moreover, different aspects of the database system may in fact use different redundancy models. For example, a database management system may have one set of processes that manage data, and another set of processes that execute queries and to which client applications connect. These sets of processes may have completely different redundancy models, and may communicate with each other in various ways.

A related topic is the partitioning scheme. In general, it is better to hide the application from the actual partitioning of the data fragments. This allows the application to remain unchanged should the partitioning scheme be changed due to planned or unplanned reconfigurations. Keeping the partitioning scheme internal to the database server allows for internal load balancing and reorganization of data without affecting applications. For performance reasons some systems provide some concept of locality and support for co-locating applications and data on the same node. This can sometimes be controlled through “hints” from the applications to the database server about where data is most effectively stored. Logical partitioning schemes for both applications and data are often combined with common load balancing schemes built into distributed communication stacks.

Products from Oracle, Sun, and MySQL maintain the process distribution transparency with various approaches.

7 Summary

Database management systems are critical components of highly available applications. To meet this need, many highly available database management systems have been developed. This paper describes the architectures that are internally used to construct these highly available databases. These architectures are examined from the perspective of both process redundancy and logical data redundancy. Process redundancy is always required; it refers to the maintenance of redundant database processes that can take over in case of failure. Data redundancy is also required. Data redundancy can be provided at either the physical or the logical level. Although both forms of data redundancy can provide high availability, this paper has concentrated on logical data redundancy since that is a case where the database explicitly manages the data copies. We believe that process and data redundancy are useful means to describe the availability characteristics of these software systems.

References

1. Application Interface Specification, SAI-AIS-A.01.01, April 2003. Service Availability Forum, available at www.saforum.org.
2. Gray, J. and Reuter, A.: Transaction Processing Systems, Concepts and Techniques. Morgan Kaufmann Publishers, 1992.
3. How MySQL Cluster Supports 99.999% Availability. MySQL Cluster white paper, MySQL AB, 2004, available at http://www.mysql.com/cluster/.
4. Hu, K., Mehrotra, S., Kaplan, S.M.: Failure Handling in an Optimized Two-Safe Approach to Maintaining Primary-Backup Systems. Symposium on Reliable Distributed Systems 1998: 161-167.
5. Humborstad, R., Sabaratnam, M., Hvasshovd, S-O., Torbjørnsen, Ø.: 1-Safe Algorithms for Symmetric Site Configurations. VLDB 1997: 316-325.
6. Hvasshovd, S., et al.: The ClustRa Telecom Database: High Availability, High Throughput and Real-time Response. VLDB 1995, pp. 469-477, September 1995.
7. Kannan, S. et al.: Configuring Highly Available Clusters Using HACMP 4.5. October 2002, available at http://www.ibm.com.
8. Norman, M.G., Zurek, T., Thanisch, P.: Much Ado About Shared-Nothing. SIGMOD Record 25(3): 16-21 (1996).
9. Maximum Availability Architecture (MAA) Overview. Oracle Corporation, 2003, available at http://otn.oracle.com/deploy/availability/htdocs/maa.htm.
10. Mohan, C., Treiber, K., Obermarck, R.: Algorithms for the Management of Remote Backup Data Bases for Disaster Recovery. ICDE 1993: 511-518.
11. MySQL Reference Manual. MySQL AB, 2004, available at http://www.mysql.com/documentation/.
12. Oracle Data Guard Overview. Oracle Corporation, 2003, available at http://otn.oracle.com/deploy/availability/htdocs/DROverview.html.
13. Oracle Real Application Clusters (RAC) Overview, Oracle Corporation, 2003, available at http://otn.oracle.com/products/database/clustering/.
14. Polyzois, C.A., Garcia-Molin, H.: Evaluation of Remote Backup Algorithms for Transaction-Processing Systems. ACM Trans. Database Syst. 19(3): 423-449 (1994).

15. SAP Standard Applications Benchmarks, SD Parallel, June 2002, available at http://www.sap.com/benchmark/.
16. Solid High Availability User Guide, Version 4.1, Solid Information Technology, February 2004, available at http://www.solidtech.com.
17. Sun Cluster 3.1 10/03 Concepts Guide, Sun Microsystems, Part No. 817-0519-10, October 2003.
18. Sun ONE Application Server 7 Enterprise Edition – Administrator's Guide, Sun Microsystems, Part no. 817-1881-10, September 2003.
19. Thalmann, L. and Ronström, M.: High Availability features of MySQL Cluster. MySQL Cluster white paper, MySQL AB, 2004, available at http://www.mysql.com/cluster/.
20. Transaction Processing Performance Council TPC-C Benchmarks, December 2003, available at: http://www.tpc.org.
21. Wiesmann, M., Schiper, A.: Beyond 1-Safety and 2-Safety for Replicated Databases: Group-Safety. EDBT 2004: 165-182.