# Replication of Data in Database Systems for Backup and Failover – An Overview

Tarandeep Singh, Parvinder S. Sandhu, and Harbax Singh Bhatti

***Abstract*—The production database are transactions intensive transactions can be insert, update on the tables, failover is the replica of the production server, if there is any change we have to implement on the production and it will be automatically implemented on failover or standby database. Now a days the data on the production server is increasing and we need extra storage space on production server to keep data and this is same required on the failover. To generate reports from that data will increase load on the production server and can affect the performance of the server. There are also some threats which can cause loss of data from which we have to protect our database like Hardware failure, loss of machine. Replication is one of the methods for Backup of the running database and its immediate failover during failure. This paper represents some parts for the replication techniques, failover, and transaction states.**

***Index Terms*—Asynchronous, synchronous, active, commit, rollback, RAIDb.**

## I. Introduction

Demand for high performance combined with plummeting hardware prices have led to the widespread emergence of large computing clusters [1]. Database systems are a key component of the computer infrastructure of most organizations. It is thus crucial to ensure that database systems work correctly and continuously even in the presence of a variety of unexpected events. The key to ensuring high availability of database systems is to use replication [2]. In this paper, we propose an approach for Replication of data for Backup and failover.

Traditional database replication techniques as provided by both commercial and open source database management system (such as Oracle, IBM DB2, Microsoft SQL Server, PostgreSQL, or MySQL) can be applied [3].

The traditional approach to replication management involves read one copy; write all copies (ROWA). These are classified by three design criteria; whether an approach is based on changes to the database kernel or by middleware that uses (nearly) unmodified single node DBMS engines for the replicas; whether updates are allowed at any site, or only at a "master" primary site, and whether transaction propagation to other replicas is done eagerly (as part of commit processing) or lazily after the commit. Multi-master and eager replication protocol, which makes use of an available group communication system to guarantee a total order delivery which allows sites to remain consistent (so, 1-copy serializability can be ensured). Improved performance came from doing the database processing at one site only, and applying efficient primary-keyed write operations at other sites, with a certification to prevent conflicts, that was done consistently at all sites. Using sites that provide SI as concurrency control, and giving one-copy SI rather than serializability, allows a great improvement in performance [4].

Manuscript received January 15, 2013; revised March 27, 2013.

Tarandeep Singh is with the Punjab Technical University, India (e-mail: tarandeepskhs@gmail.com).

Parvinder S. Sandhu is with the Rayat and Bahra Institute of Engineering and Bio-Technology, Mohali, Punjab, India (e-mail: parvinder.sandhu@gmail.com).

Harbax Singh Bhatti is with the Department of Applied Sciences, Baba Banda Singh Bahadur, Engineering College, Fatehgarh Sahib (India).

## II. Data Replication Types

### A. Synchronous Replication

When synchronous replication is applied, a change made to a data at the primary site is synchronously replicated to a data volume at a secondary site. This ensures that the secondary site has an identical copy of the data at all times. Write I/O operation acknowledgement is sent to the application only after the write I/O operation is acknowledged by the storage subsystem at both the primary and the secondary site. Before responding to the application, the storage subsystem must wait for the secondary subsystem I/O process to complete, resulting in an increased response time to the application. Thus, performance with synchronous replication is highly impacted by factors such as link latency and link bandwidth. Deployment is only practical when the secondary site is located close to the primary site. When evaluating the use of synchronous replication, an important consideration is the behavior of the storage subsystem when the connection between the primary and secondary subsystem is temporarily disrupted [5]. Synchronous replication does not provide protection against data corruption and loss of data due to human errors [5]. Snapshot technology must be used with synchronous replication to provide full protection against both losses of access to data and loss of data due to data corruption [5].

### B. Asynchronous Replication

In an asynchronous mode of operation, I/O operations are written to the primary storage system and then sent to one or more remote storage systems at some later point in time. Due to the time lag, data on the remote systems is not always an exact mirror of the data at the primary site. This mode is ideal for disk-to-disk backup or taking a snapshot of data for offline processes, such as testing or business planning. The time lag enables data to be replicated over lower-bandwidth networks, but it does not provide the same level of protection as synchronous replication. Asynchronous replication is less sensitive to distance and link transmission speeds [5]. However, because replication might be delayed, data can be lost if a communication failure occurs followed by a primary site outage. According to deferred update technique, transactions are processed locally at one database server and when committed, are forwarded for certification to the other servers. Deferred update replication offers many advantages over its alternative, immediate update replication, which synchronizes every transaction operation across all servers [5].

The advantages and disadvantages, one may cite:

### C. *Advantages*

- SAN network-based replication is storage agnostic and server-architecture agnostic [5]. When combined with mirroring or snapshot functionality, SAN network-based replication can be easily integrated into and managed in an application/database environment [5].
- Better performance, by gathering and propagating multiple updates together, and localizing the execution at a single, possibly nearby, server (thus reducing the number of messages in the network) [6].
- Better support for fault tolerance, by simplifying server recovery (i.e., missing updates may be treated by the communication module as lost messages) [6].
- Lower deadlock rate, by eliminating distributed deadlocks [6].

### D. *Drawbacks*

- Replication is limited to a single pair of servers or, at best, several servers running the same version of operating system. An additional cost or license might be required for each server added [5].
- SAN network-based replication adds latency to the primary data path. When an application writes data to primary storage, the data is also written to the remote site, adding delay to the write operation. Thus, the SAN infrastructure does not act as a transparent layer between the server and the primary storage, which can make diagnosing faults more difficult [5].
- The disadvantage of synchronous replication is that a write I/O operation acknowledgement is sent to the application only after the write I/O operation is acknowledged by the storage subsystem at both the primary and the secondary site. Before responding to the application, the storage subsystem must wait for the secondary subsystem I/O process to complete, resulting in an increased response time to the application. Thus, performance with synchronous replication is highly impacted by factors such as link latency and link bandwidth. Deployment is only practical when the secondary site is located close to the primary site [5].
- The main drawback of the deferred update technique is that the lack of synchronization during transaction execution may lead to large transaction abort rates [6].

## III. Transaction States

During its processing, a transaction passes through some well-defined states. The transaction starts in the executing state, when it’s read and writes operations are locally executed at the database site where it was initiated. When the client that initiates the transaction requests the commit, the transaction passes to the committing state and is sent to the other database sites. A transaction received by a database site is also in the committing state, and it remains in the committing state until its fate is known by the database site (i.e., commit or abort) [3]. Different states are shown with the help of following Fig. 1.

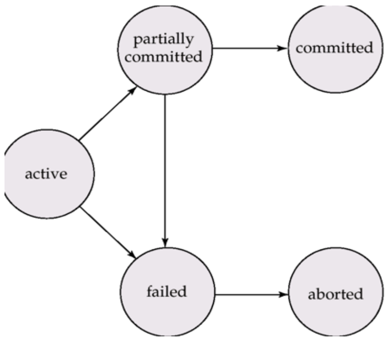

Fig. 1. Example of transaction states

### A. *Active State*

It is divided into two phases.

- Initial Phase: A database transaction is in this phase while its statements start to be executed [7], [8].
- Partially Committed Phase: A database transaction enters this phase when its final statement has been executed [7], [8]. At this phase, the database transaction has finished its execution, but it is still possible for the transaction to be aborted because the output from the execution may remain residing temporarily in main memory - an event like hardware failure may erase the output. [7], [8].

### B. *Failed State*

A database transaction enters the failed state when its normal execution can no longer proceed due to hardware or program errors [7], [8].

### C. *Aborted State*

A database transaction, if determined by the DBMS to have failed, enters the aborted state. An aborted transaction must have no effect on the database, and thus any changes it made to the database have to be undone, or in technical terms, rolled back. The database will return to its consistent state when the aborted transaction has been rolled back. The DBMS's recovery scheme is responsible to manage transaction aborts [7], [8].

### D. *Committed State*

A database transaction enters the committed state when enough information has been written to disk after completing its execution with success [7], [8]. In this state, so much information has been written to disk that the effects produced by the transaction cannot be undone via aborting; even when a system failure occurs, the changes made by the committed transaction can be re-created when the system restarts [7], [8].

## IV. Failover

Failover is one basic fault-tolerance function of mission-critical systems that are providing a constantly accessible service. Its purpose is to redirect requests from the failing system to a backup that mimics the operations of the first one. The whole process is supposed to happen automatically and transparently to the end user. Failover or switchover solutions are widely used whenever and wherever high availability is needed. Adopting such an approach is neither new nor original; yet, the real challenges of this work are its wide scope, the number of teams involved, and the geographically distributed nature of both the Grid and the related operational tools [9].

The failover concern played a great role in the way this architecture has been designed: the clear separation between the different modules implies indeed an easier replication work, as well as many possibilities of failover scenarios. These scenarios include partial switches, each of these modules being able to work with any of the replicas of the other modules. The failover process consists of three phases: the first step is to detect failure, the second step to identify and activate the standby resource, and the last step for the standby application to go active [9].

Data can be replicated automatically and precisely to many locations. However replication works as a defense if we use logical replication over distinct database systems. Many replication algorithms copy data values from the source data item to its replicas. Logical replication copies the command that caused the source data item to change. The command is executed at each replica's site and, because of one copy serializability, results in the same new value for the replica. If we assume a distinct provenance (defined in the next section) for the database system software at each site, then the Trojan horse will not be replicated at all sites. An attack must compromise multiple, possibly heterogeneous, host programs, an unlikely event in practical systems. Even if the attackers can succeed at every site, the attack still may fail. Other Threat to the database system are Manual attacks are carried out by giving malicious commands to the database system. An n-person rule requires n humans outside the system to agree to a change to the database. Transaction control expressions are a more general form of this concept. They require multiple users to agree to specific conditions defined on specific steps of a transaction [10].

## V. Redundant Array of Inexpensive Databases

One of the goals of RAIDb is to hide the distribution complexity and provide the database clients with the view of a single database like in a centralized architecture [11], [12].

RAIDb controllers can offer caching to hold the replies to SQL queries. The controller is responsible for the granularity and the coherence of the cache. Additional features such as connection pooling can be provided to further enhance performance scalability. There is no restriction to the set of services implemented in the RAIDb controller. Monitoring, debugging, logging or security management services can prove to be useful for certain users [11], [12].

Three basic RAIDb levels varying the degree of partitioning and replication among the databases. RAIDb-0 (database partitioning) and RAIDb-1 (database mirroring) are similar to RAID-0 (disk striping) and RAID-1 (disk mirroring), respectively. Like RAID-5, RAIDb-2 is a tradeoff between RAIDb-0 and RAIDb-1. Actually, RAIDb-2 offers partial replication of the database [11], [12].

### A. *RAIDb-0*

RAIDb-0 consists in partitioning the database tables among the database backend nodes. A table itself cannot be partitioned but the different tables can be distributed on different backend nodes. RAIDb-0 requires at least two database backend, provides moderate performance scalability but does not offer fault tolerance [11], [12]. Fig. 2, "RAIDb-0 example" shows an example of a RAIDb-0 configuration [11], [12].

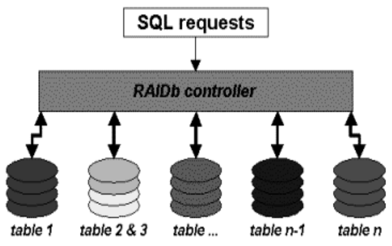

Fig. 2. RAIDb-0 overview

### B. *RAIDb-1*

RAIDb-1 offers a full mirroring or full replication of the database on the backend. It offers the best fault tolerance scheme since the system is still available with only one backend. On the minus side, there is no speedup on writes (UPDATE, INSERT, DELETE requests) since they have to be broadcasted to all nodes. Fig. 3, "RAIDb-1 example" shows an example of a RAIDb-1 configuration [11], [12].

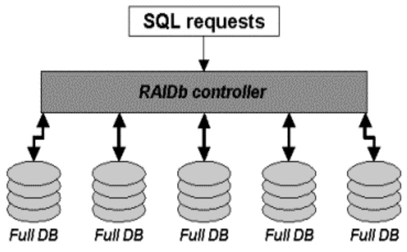

Fig. 3. RAIDb-1 overview

### C. *RAIDb-2*

RAIDb-2 is a tradeoff between RAIDb-0 and RAIDb-1. It provides partial replication to tune the degree of replication of each database table to obtain the best read/write throughput. RAIDb-2 requires that each database table is available on at least two nodes. Fig. 4, "RAIDb-2 example" shows an example of a RAIDb-2 configuration [11], [12].

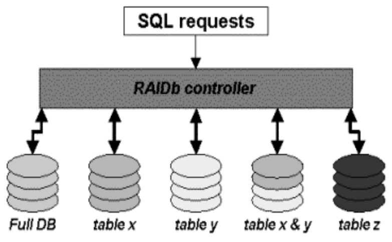

Fig. 4. RAIDb-2 overview

### D. Nested RAIDb Levels

It is possible to compose several RAIDb levels to build large scale configurations or meet specific needs. The next example is a RAIDb-1-0 configuration where a top level RAIDb-1 controller dispatches the requests to three full databases implemented with a RAIDb-0 controller. Fig. 5, “RAIDb-1-0 example” shows an example of a RAIDb-1-0 configuration [11], [12].

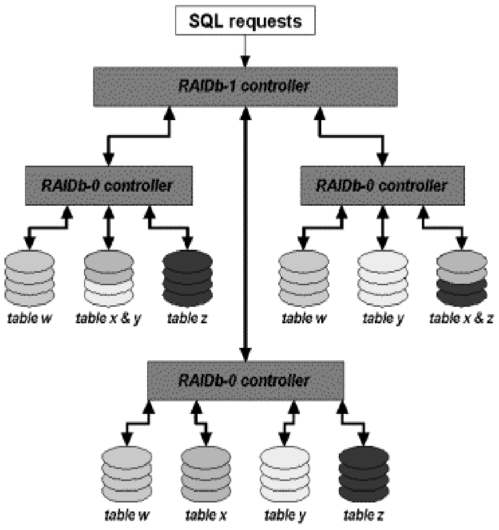

Fig. 5. Example of a RAIDb-1-0 composition

### E. RAIDb-0-1

This last example (Fig. 6, “RAIDb-0-1 example”) shows a RAIDb-0-1 composition. The top level is a RAIDb-0 controller and fault tolerance is achieved on each partition using a RAIDb-1 controller [12].

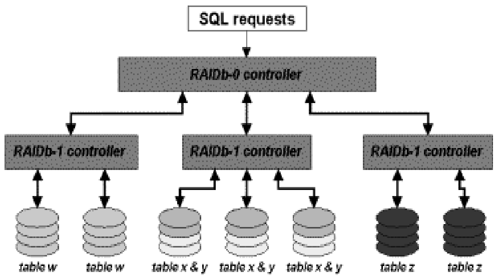

Fig. 6. Example of a RAIDb-0-1 composition

## VI. Conclusion

Applications need high scalability and availability at low and controlled cost. Utility computing is not limited only to the single database system for support and high performance. The purpose of this paper is to show methods of replication and failover for the performance upgrade for the mission critical and transaction intensive databases.

## References

[1] L. Camargos, F. Pedone, and R. Schmidt, “A primary-backup protocol for in-memory database replication,” in *Proc. Fifth IEEE International Symposium on Network Computing and Applications*, pp. 204-211, July 24-26, 2006.

[2] R. Garcia, R. Rodrigues, and N. Pregui ça, “Efficient middleware for byzantine fault tolerant database replication,” in *EuroSys '11 Proc. the sixth Conference on Computer systems*, pp. 107-122, ACM New York, NY, USA, 2011

[3] J. Osrael, L. Froihofer, M. Weghofer, and K. M. Goeschka, “Axis2-based replication middleware for web services,” in *Proc. IEEE International Conference* on *Web Services, ICWS,* pp. 591-598, July 9-13, 2007.

[4] H. Jungy, H. Han, A. Fekete, and U. Rohm, “Serializable snapshot isolation for replicated databases in high-update scenarios,” presented at the VLDB Endowment, Seattle, Washington, August, 2011.

[5] P. Brouwer, “The art of data replication,” An Oracle Technical White Paper, Oracle Corporation World Headquarters 500 Oracle Parkway Redwood Shores, CA 94065 U.S.A, pp. 1-28, September 2011.

[6] F. Pedone, R. Guerraoui, and A. Schiper, “The database state machine approach,” in *Manufactured in The Netherlands Distributed and Parallel Databases*, Kluwer Academic Publishers, pp. 71–98, 2003.

[7] Transaction states. [Online]. Available: http://www.jkinfoline.com/transaction-states.html

[8] Database transaction. [Online]. Available: http://it.toolbox.com/wiki/index.php/Database_Transaction

[9] A. Cavalli, A. Pagano, O. Aidel, C. L’orphelin, G. Mathieu, and R. Lichwala, “Geographical failover for the EGEE-WLCG grid collaboration tools,” in *Proc. International Conference on Computing in High Energy and Nuclear Physics*, vol. 119, 2008.

[10] J. P. McDermott, “Replication does survive information warfare attacks,” in *Proc. the IFIP TC11 WG11.3 Eleventh International Conference on Database Security XI 1998*, pp. 219-228, London, UK, UK, 1998.

[11] E. Cecchet, J. Marguerite, and W. Zwaenepoel, “RAIDb: Redundant array of inexpensive databases,” Institut National De Recherche En Informatique Et En Automatique, September, 2003.

[12] RAIDb basics. [Online]. Available: http://c-jdbc.ow2.org/current/doc/userGuide/html/ar01s10.html

**Tarandeep Singh** is a Master in Computer Applications (MCA) doing Doctorate from Punjab Technical University, Punjab, India. And is working as a Sr. Lecturer in Gyan Jyoti Institute of Management and Technology, Mohali, Punjab, INDIA. Email: tarandeepskhs@gmail.com

**Parvinder S. Sandhu** obtained his doctorate in Computer Science and Engineering and is currently working as Professor in Computer Science & Engineering department at Rayat & Bahra Institute of Engineering and Bio-Technology, Mohali, Punjab, INDIA. He is editorial committee member of various International Journals and conferences. He has published more than 150 research papers in various referred International journals and conferences. He chaired more than 100 renowned International Conferences and also acted as keynote speaker in different countries. His current research interests are Software Reusability, Software Maintenance, Machine Learning and Image Processing. Email: parvinder.sandhu@gmail.com

**Harbax Singh Bhatti** received his Ph.D in Mathematical Modeling, Partial Differential Equations and is currently working as Professor and HOD (Applied Sciences) in Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib, Punjab (India).