

# Replication Methods and Their Properties

Lars Frank

Department of Informatics, Copenhagen Business School

## ABSTRACT

This article will use the latest research within extended transaction models and replication methods to illustrate how to optimize systems by using replicated data. In general, it is not possible to select one method as the best because the methods have very different properties. The most important evaluation criteria for replication methods are availability, performance, consistency and costs. However, these evaluation criteria have to be subdivided to illustrate the different properties of the different methods.

*Keywords:* Replication methods, ACID properties, availability, fault tolerance, multidatabases.

## 1. INTRODUCTION

Replication methods may be used to improve performance and response time by substituting remote data accesses with local data accesses. The availability of the system can be increased by using replicated data in case a local failure/disaster should occur. The major disadvantages of data replication are the additional costs of updating replicated data and the problems related to managing the consistency of the replicated data. Tables 1 and 2 give an overview of the evaluation of the replication methods described in this article. Frank, 1999, describes how such replication overviews may be used to optimize databases in practice. However, Frank, 2006, evaluates many more DBMS replication methods and, therefore, it is possible to optimize even more. It is also important to compare the replication designs with the no-replication design as this design is most used in practice.

The article is organized as follows: Section 2 will describe the evaluation criteria used. Section 3 will describe the most important replication methods used in practice. Arguments for the evaluation illustrated in Tables 1 and 2 are presented in both sections 2 and 3. Concluding remarks are presented in section 4.

Related Research: Different versions of the 1-safe and 2-safe designs have been described in e.g. Garcia-Molina and Polyzois, 1990, Polyzois and Garcia-Molina, 1994, Gellersdörfer and Nicola, 1995 and Humborstad et al., 1997. The “0-safe design with local commit” has been used in practice for a number of years and is described by Frank and Zahle, 1998. The 2-safe design, the basic 1-safe design, “the 1-safe design with commutative updates” and “the 0-safe design with local commit” have all been described and evaluated in detail by Frank, 1999. A more up to date evaluation of the replication designs is found in Frank, 2006. The *countermeasure transaction model* is the first transaction model that systematically tries to describe how to eliminate or reduce the consistency and anomaly problems, which some replication methods may cause. The model was first described by Frank and Zahle, 1998, but a much more extensive description may be found in Frank, 2006. This transaction model owes many of its properties to e.g. Garcia-Molina and Salem, 1987; Mehrotra et al., 1992; Weikum and Schek, 1992 and Zhang, 1994. Many new versions of the classic replication designs have been described in recent years. The most important of these are described after the corresponding classic design.

**Table 1. Evaluation overview of some of the most important table replication designs.**

Properties	DBMS supported replication methods						
	n-safe design	Quorum-safe design	1-safe design. Basic solution	1-safe design with commutative updates	0-safe design with local commit	0-safe designs with deferred commit	No-replication design
Read performance/capacity	Best	Worst	Average	Average	Best	Best	Average
Write performance	Worst	Above worst	Average	Average	Best	Below best	Average
Ease of failure recovery	Average	Average	Worst	Average	Best	Best	Average
Ease of disaster recovery	Best	Below best	Above worst	Average	Average	Average	Worst
The probability of lost data <sup>1</sup>	Best. $p^n$	Below best $p^{\lfloor n/2 \rfloor}$	Worst $p$	Average	Average	Average	Worst $p$
Logging of the update transaction <sup>2</sup>	Not supported	Not supported	Not supported	Recommended	Recommended	Recommended	Not supported
Availability <sup>3</sup>	$1-q^n$	$1-q^n$	$1-q^n$	$1-q^n$	$1-q^n$	$1-q^n$	$1-q$
Atomicity	Best	Best	Worst	Best	Best	Best	Best
Consistency	Best	Best	Average	Average	Worst	Worst	Best
Isolation	Best	Best	Average	Average	Worst	Worst	Best
Durability	Best	Best	Worst	Best	Best	Best	Best
Development costs	Best	Best	Best	Average	Worst	Worst	Best

<sup>1</sup> The probability of lost data as a function of the probability, say  $p$ , of a local disaster.

<sup>2</sup> Logging of the update transactions in the locations of the clients.

<sup>3</sup> The availability as a function of the probability, say  $q$ , of a local site failure.

**Table 2. Evaluation overview of operating system replication methods**

Evaluation criteria	Operating system replication methods			
	Mirroring with disk volume ownership	Mirroring without disk volume ownership	Remote caching in a fast storage	Local caching in the user location
<b>Read performance</b>	Average	Best	Average	Best
<b>Write performance</b>	Average	Above worst	Average	Average
<b>Ease of failure recovery</b>	Average	Average for roll back recovery	Average	Not supported
<b>Ease of disaster recovery</b>	Below best	Below best	Not supported	Not supported
<b>The probability of lost data</b>	Best $p^n$	Best $p^n$	Worst	Worst
<b>Availability</b>	$1-q^n$	$1-q^n$	$1-q^2$	$1-q^2$
<b>Atomicity</b>	Best	Not supported	Best	Not supported
<b>Consistency</b>	Best	Not supported	Best	Not supported
<b>Isolation</b>	Best	Not supported	Best	Not supported
<b>Durability</b>	Best	Not supported	Best	Not supported
<b>Development costs</b>	Best	Best	Best	Best

## 2. DESCRIPTION OF OUR EVALUATION CRITERIA FOR REPLICATION METHODS

In this article, the properties of the replication designs are normally described as Best, Average, Worst, Below average, etc., which allows us to compare the designs relatively. In general, it is not possible to select one of the replication designs as the best because all the designs have very different properties. However, if the requirements of an application are known, it is possible to select the most inexpensive design that fulfills the needs of the application. Below, is given a short description of the evaluation criteria used in this article.

The *read performance* property of a table design is evaluated to be “best” if remote readings always can be substituted by local readings. The “n-safe design with the quorum protocol” has the “worst” read performance because it is always necessary to lock a majority of the record copies to implement the isolation property.

The *write performance* property of a table design is evaluated to be “best” if a global table update always can be committed locally without communication with other locations. The “n-safe design with the ROWA protocol” is evaluated to be “worst” in terms of write performance because it is always necessary to lock all the record copies to implement the isolation property.

After a local failure, the local database and its log files are not physically destroyed, and in contrast to database disasters it is always possible to repair the site in such a way that

operation may continue by using the failed site. *Ease of failure recovery* is evaluated to be “best” if the system automatically can make recovery without aborting all non-committed transactions in case of a site failure. *Ease of failure recovery* is evaluated to be “average” if the system always has to abort non-committed transactions in case of a site failure. The basic 1-safe design is evaluated to have the “worst” ease of failure recovery because even committed transactions may be lost after a failure where a secondary copy is used as the primary copy (*lost transactions*).

A *database disaster* is as a situation where a local database and its log files are destroyed. *Ease of disaster recovery* is evaluated to be “best” if it is possible to repair the database automatically. The no replication design is evaluated to be “worst” in terms of ease of disaster recovery because at best only an old remote database copy can be used for recovery.

In case of a disaster, data may be lost. The probability of avoiding lost data is evaluated to be better, the lower the probability is.

In case of a disaster, committed transactions may be lost except in the n-safe designs. Therefore, the evaluation criteria, *ease of disaster recovery* and *availability*, depend on the evaluation criterion *logging of the update transactions in the locations of the clients*.

The *availability* of a database can be defined as the probability of having access to the database.

An update transaction is *atomic* if all or none of its updates are executed. The *atomicity* property of a table design is evaluated to be “best” if the DBMSs and the transaction model can guarantee the property even in case of failure. The basic 1-safe design is evaluated to be “worst” in terms of the atomicity property because lost transactions may occur after the commitment of a transaction.

A database is *consistent* if the data in the database complies with some user-defined consistency rules. Transactions have by definition the *consistency property* if they fulfill the following condition:

“If a database is consistent when a transaction starts, the database must also be consistent when the transaction has been committed”.

The consistency property of a table design is evaluated to be “best” if the DBMSs and the transaction model can guarantee the property even in case of failure. The 0-safe designs are evaluated as “worst” in terms of consistency property because in these designs the distributed database is normally inconsistent, and, therefore, only “asymptotic consistency” can be implemented, i.e. the database converges towards a consistent state.

Transactions are executed in *isolation* if the updates of each transaction only can be seen by other transactions after the updates have been committed or aborted. The isolation property of a table design is evaluated to be “best” if the DBMSs and the transaction model can guarantee the property even in case of failure. The 0-safe designs are evaluated to be “worst” in terms of isolation property because all types of isolation anomalies may occur if they are not managed by adopting countermeasures (Frank and Zahle, 1998).

The updates of transactions are said to be *durable* if they are stored in stable storage and secured by a log recovery system. The durability property of a table design is evaluated to be “best” if the DBMSs and the transaction model can guarantee the property even in case of failure. The basic 1-safe design is evaluated to be “worst” in terms of the durability property because lost transactions may occur after the commitment of a transaction, i.e. the updates are not durable in case of a failure where a secondary copy is used as the primary copy.

The *development costs* of a table design are evaluated to be “best” if no special application programming is needed, i.e. all replication problems are managed by using DBMS tools. Therefore, the development costs of the n-safe designs, the no-replication design and the basic 1-safe design have the “best” rating in table 1. The development costs of the 0-safe designs are evaluated as “worst” because in these designs one has to select and implement countermeasures against both the lost update anomaly, the dirty read anomaly, etc (Berenson et al., 1995 and Breibart, 1992). The development costs of “the 1-safe design with commutative updates” are evaluated as “average” because in this design it is only necessary to implement countermeasures against the lost update anomaly.

The replication costs of the “1-safe design with commutative updates” and the 0-safe designs may be high. Therefore, these designs are best suited for large computer systems or application software packages such as ERP and CSCW systems where it is possible to implement the replication as part of the software package.

### **3. DESCRIPTION OF THE MOST IMPORTANT REPLICATION METHODS**

In Table 1, the replication methods described by Frank, 1999, are evaluated together with the main types of the n-safe design and a group of 0-safe designs that have deferred commit. This article will also evaluate replication designs that can be implemented as part of the operating system, and, therefore, cannot automatically support ACID properties.

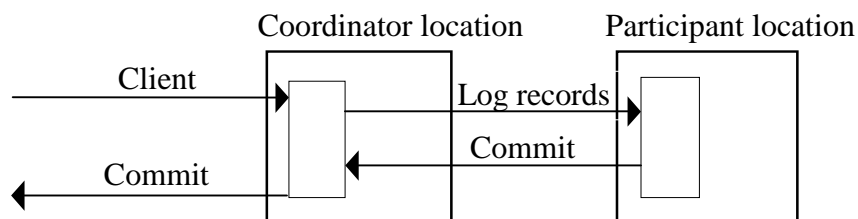
All the n-safe designs described in this article can be optimized by using the so called *atomic broadcast protocol* (Hadzilacos and Toueg, 1993 and Weismann et al. 2000). This is a group communication technique where messages from the coordinator are delivered in the same order in all the participating locations. This property can simplify the distributed concurrency control if the coordinator uses the atomic broadcast protocol to transfer the execution order in the coordinator location to the locations of the participants. This is possible if data is replicated in such a way that the coordinator can execute transactions locally and only at the end uses the atomic broadcast protocol to coordinate the update of replicated data. At the same time, performance is improved as there is communication between the involved locations once per transaction and not once per data access. However, the improved performance possibilities are not reflected in table 1 as the atomic broadcast protocol has not been implemented in any commercial DBMS product. All the n-safe designs described above have *eager replication*, i.e. the updates of transactions are propagated to any replicated copies before the commit of the transactions.

Performance of the 1-safe and 0-safe designs can be optimized by *lazy replication*, where the updates to any replicated copies are propagated after the commitment of the updating transaction. This improves the response time at the costs of the consistency between replicated data. The atomic broadcast protocol described above cannot improve the performance of the lazy replication designs as the transactions have already been committed when the update of

replicated data takes place. However, the performance of the 1-safe designs and 0-safe designs can be improved in the same way as the n-safe designs by replicating data in such a way that transactions can run locally until commitment.

### 3.1 The n-Safe Design with the ROWA Protocol and the 2-Safe Design

The 2-safe design is a special case of the n-safe design where only two copies of a file exist. In the 2-safe design (Gray and Reuter, 1993), the primary/coordinating transaction manager involves the backup/participant system in the commit. If the participant system is up, the coordinating transaction manager sends the log records of the transaction at the end of commit phase 1. The coordinating transaction manager will not commit until the participant responds (or is declared down). The 2-safe design has good ratings in read capacity, consistency, isolation and easiness of failure/disaster recovery. The main problem with the solution is the poor write performance and a high risk of transaction restarts in case of major failures in one of the database locations or in the network connecting the locations. The 2-safe design may easily be generalized to the n-safe design where  $n > 2$ . In the n-safe design, the coordinating transaction manager commits an update as a function of the n-1 participating transaction managers' responses. There are many versions of the n-safe design. However, in all the versions it is possible to optimize the write performance if the participating transaction managers answer immediately without first forcing the log records to durable storage. This is acceptable in the n-safe design because in case of a disaster, it is extremely unlikely that all n copies of the log are involved in the disaster. Anyway, the n-safe design still has a relatively low write performance.



**Figure 1** 2-safe database design

ROWA (Read One, Write All) means that in terms of reading only one copy of the record is read and in terms of writing all the copies must be written/updated. The n-safe design with the ROWA protocol has the best read performance as a local record copy often can be used. The write performance is evaluated as worst, as all record copies must be locked before an update can be executed. The ROWA protocol is very vulnerable to both communication and site failures. The ROWAA (Read One, Write All Available) protocol is a new version of the ROWA protocol. It is tolerant to both communication and site failures at the costs of controlling that all participating copies are available when the transaction is committed (e.g. Coulouris et al., 2001). However, this extra control reduces the performance for all transactions. In stable networks where failures are rare, it is therefore much better to restart the traditional ROWA protocol with a new value of n in case of communication or site failures.

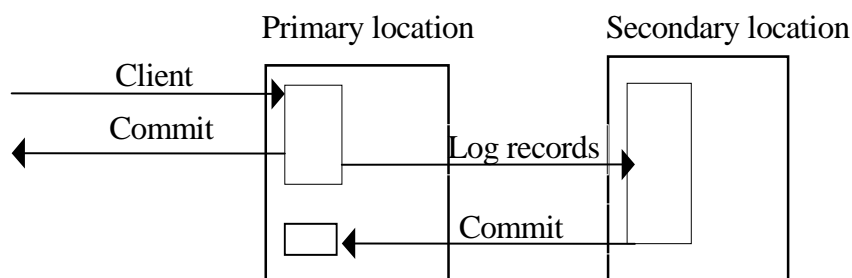
### 3.2 The n-Safe Design with the Quorum Consensus Protocol

The *quorum consensus protocol* is tolerant to both communication and site failures as only a number of locations with a 'quorum' are necessary for accessing data. In the quorum consensus protocol, a number of 'votes' is assigned to each copy. Each read operation must obtain a read quorum R before it can execute the read operation, and each write/update operation must obtain a write quorum W before it can execute the write/update. In order to

obtain the isolation property,  $R$  and  $W$  must be selected in such a way that  $W$  is greater than half the amount of votes and  $R+W$  are greater than the total number of votes. If  $R = 1$ ,  $W =$  the number of locations with a copy, and each location has one vote, the quorum consensus protocol specializes itself to the ROWA protocol, and, therefore, it is assumed in the following that  $R$  is greater than 2. In this case, the ROWA protocol has a better read performance than the quorum consensus protocol as the ROWA protocol only has to read one copy. The quorum consensus protocol has a better write performance than the ROWA protocol as the quorum consensus protocol only has to lock  $W$  copies before it is possible to write/update replicated data. The quorum consensus protocol provides better availability than the ROWA protocol as the quorum consensus protocol can tolerate some communication and site failures.

### 3.3 The Basic 1-Safe Design

In the basic 1-safe design (Gray and Reuter, 1993), the primary transaction manager goes through the standard commit logic and declares completion when the commit record has been written to the local log. In a 1-safe design, the log records are asynchronously spooled to the locations of the secondary copies. In case of a primary site failure in the 1-safe design, production may continue by selecting one of the secondary copies as a new primary copy. When the failure in the old primary copy location has been repaired, the log records from this location cannot always be used to update the new primary location (former secondary copy) because the records in the new primary copy may have been updated. *Lost transactions* are defined as the updates committed in the failed old primary copy and not in the new primary copy at the time production restarts in the new primary location. The main problem with the 1-safe design is that the lost transactions must be reconstructed and re-executed before the recovery process is finished. Discontinuing production after a failure may prevent this problem. Therefore, selection of a new primary copy is only used in case of a disaster or a very serious failure.

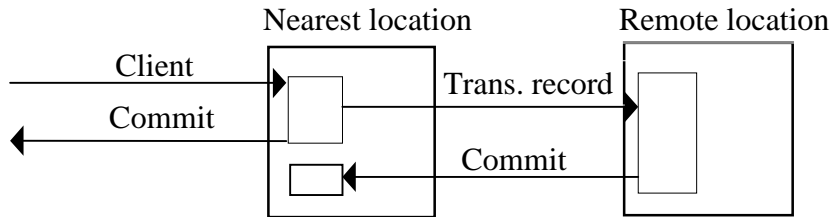


**Figure 2** 1-safe database design

### 3.4 The 0-Safe Design with Local Commit

The *0-safe design with local commit* is defined as  $n$  table copies in different locations where each transaction first will go to the nearest database location, where it is executed and committed locally. If the transaction is an update transaction, the transaction propagates asynchronously to the other database locations, where the transaction is re-executed without user dialog and committed locally at each location. This means that all the table copies normally are inconsistent and not up to date under normal operation. The inconsistency must be managed by using countermeasures against the isolation anomalies. For example, to prevent lost updates in the 0-safe design, all update transactions must be designed to be commutative (Frank and Zahle, 1998). From a performance and disconnection point of view, the 0-safe design with local commit is the best choice, and, therefore, it is recommended for

mobile computing. Other versions of the 0-safe design where commitment is deferred are described later.



**Figure 3** 0-safe database design

### 3.5 The 1-Safe Design with Commutative Updates

The 1-safe design can transfer updates to the secondary copies in the same way as the 0-safe design. In this design, lost transactions cannot occur because the transaction transfer of the 0-safe design makes the updates commutative. The properties of this mixed design will come from either the 1-safe or the 0-safe design. The 1-safe design with commutative updates does not have the high update performance and capacity of the 0-safe design. On the other hand, in this design the isolation property may be implemented automatically as long as the primary copy does not fail. Normally, this makes it much cheaper to implement countermeasures against the isolation anomalies, because it is only necessary to secure that “the lost transactions” are not lost in case of a primary copy failure.

### 3.6 The 0-Safe Designs with Deferred Commit

The “*0-safe designs with deferred commit*” are defined as a group of replication methods where a global transaction first must update the local copy closest to the user by using a compensatable subtransaction. Later, the local update may be committed globally by using one of the following designs:

- The “*0-safe design with primary copy commit*” where the global transaction may be committed/rejected in a (remote) primary copy location. If the update of the primary copy is committed, retrievable subtransactions will be propagated to the rest of the secondary copy locations, which must be updated and committed sooner or later. However, if the update of the primary copy is rejected, a compensating subtransaction must be propagated to the location where the transaction was first created.
- The “*0-safe design with all participants commit*” where the global transaction may be committed if all the participating locations with local copies can accept the original compensatable updating.
- The “*0-safe design with token commit*” where the global transaction may be committed when the user who initiated the compensatable subtransaction receives a “token” that excludes all other users from committing global transactions.

### 3.7 The No-replication Design

The *no-replication design* is by definition without on-line replication. For recovery reasons, it is important to store remote off-line copies.

### 3.8 Mirroring

*Mirroring* techniques enable the operating system (eventually supported by hardware) to write the data simultaneously on different disks in such a way that the data is replicated two or more times as needed. Mirroring methods may be grouped in the same way as the n-safe, 1-safe and 0-safe designs. However, this article only evaluates n-safe mirroring methods as these methods have the most interesting properties compared to the corresponding n-safe designs. Mirroring replication methods can also be grouped according to the processes that can access data under normal operation. In mirroring with disk volume ownership, a single concurrency managing process owns a disk volume and its mirrored data as long as the process is operative. Therefore, read performance is average as all reads must go through this process. However, in this situation it is also possible to optimize write performance to average, as distributed locking is not necessary. Failure recovery is the same as in a central database except that a database copy may fail without any problems. Therefore, it does not matter that mirroring does not support ACID properties in this situation. In mirroring without disk volume ownership, concurrent processes with their own local DBMS may lock data in a ROWA like fashion, and, therefore, the read performance is best and the write performance close to worst.

### 3.9 Caching

*Cached data* is a snapshot of some frequently used data. Normally, cached data is incomplete, and, therefore, it is not suited for disaster recovery. In *remote caching*, a primary copy of the frequently used data is normally stored on a very fast medium to optimize access to data. In *local caching*, an often inconsistent secondary copy of the frequently used data is stored in or close to the location of some users to optimize their access to the data. The local cache may be updated periodically or when a local user makes an update. Therefore, the local cache is often inconsistent.

## 4. CONCLUSIONS

To our knowledge, this article is the first to evaluate all the most commonly used replication methods. The replication methods are grouped into types, and two methods within each group are evaluated. Tables may be fragmented and replication can be optimized by selecting the best replication method for each fragment. However, inconsistencies and anomalies may complicate the selection of replication methods as countermeasures against anomalies should be selected per application/system and not per fragment as described by Frank and Zahle, 1998.

Hybrid replication protocols (see e.g. Irún et al., 2003) can be developed in such a way that they behave as different replication methods, depending on the system configuration or the application requirements. However, such solutions are not yet available in commercial database management systems.

## REFERENCES

- Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E. and O'Neil, P., 1995, "A Critique of ANSI SQL Isolation Levels", *Proc ACM SIGMOD Conf.*, pp. 1-10.
- Coulouris, G., Dollimore, J. and Kindberg, T. (2001), *Distributed Systems Concepts and Design*, Addison Wesley.
- Breibart, Y., Garcia-Molina, H. and Silberschatz, A., 1992, "Overview of Multidatabase Transaction Management", *VLDB Journal*, 2, pp 181-239.
- Frank, L., 2006, Published in: *Architecture for Distributed ERP Systems, Trends in Enterprise Application Architecture, VLDB Workshop*, Editors: Dirk Draheim, Gerald Weber, Lecture Notes in Computer Science, Springer-Verlag, pp. 71 - 83

Frank, L., 1999, "Evaluation of the Basic Remote Backup and Replication Methods for High Availability Databases", *Software - Practice & Experience*, Vol. 29, issue 15, pp 1339-1353.

Frank, L., 2007, "Databases and Applications with Relaxed ACID Properties", Doctoral Thesis, *Copenhagen business School*, Accepted for defense 2007.

Frank, L. and Zahle, T., 1998, "Semantic ACID Properties in Multidatabases Using Remote Procedure Calls and Update Propagations", *Software - Practice & Experience*, Vol.28, pp77-98.

Gallersdörfer, R. and Nicola, M., 1995, "Improving Performance in Replicated Databases through Relaxed Coherency", *Proc 21st VLDB Conf*, 1995, pp 445-455.

Garcia-Molina, H. and Salem, K., 1987, "Sagas", *ACM SIGMOD Conf*, pp 249-259.

Garcia-Molina, H. and Polyzois, C., 1990, "Issues in disaster recovery", *IEEE Comcon.*, IEEE, New York, pp 573-577.

Gray, J. and Reuter, A., 1993, "Transaction Processing", *Morgan Kaufman*, 1993.

Hadzilacos, V. and Toueg, S. (1993), Fault-Tolerant Broadcasts and Related Problems. *Distributed Systems*, Sape Mullender (ed.): Addison-Wesley, pp 97-145.

Humborstad, R., Sabaratnam, M., Hvasshovd, S. and Torbjornsen, O., 1997, "1-Safe algorithms for symmetric site configurations", *Proc 23th VLDB Conf, 1997*, pp. 316-325.

Irún, L., Muñoz, F. D. and Bernabéu, J. (2003), An Improved Optimistic and Fault-Tolerant Replication Protocol, *Lecture Notes in Computer Science*, 2822, pp.188-200.

Mehrotra, S., Rastogi, R., Korth, H., and Silberschatz, A., 1992, "A transaction model for multi-database systems", *Proc International Conference on Distributed Computing Systems*, pp. 56-63.

Polyzois, C. and Garcia-Molina, H., 1994, "Evaluation of Remote Backup Algorithms for Transaction-Processing Systems", *ACM TODS*, 19(3), pp. 423-449.

Weikum, G. and Schek, H., 1992, "Concepts and Applications of Multilevel Transactions and Open Nested Transactions", *A. Elmagarmid (ed.): Database Transaction Models for Advanced Applications*, Morgan Kaufmann, pp. 515-553.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B. and Alonso, G. (2000), Understanding Replication in Databases and Distributed Systems. In *Proceedings 20<sup>th</sup> International Conference on Distributed Computing Systems (ICDCS'2000)*, Taipei, Republic of China, IEEE.

Zhang, A., Nodine, M., Bhargava, B. and Bukhres, O., 1994, "Ensuring Relaxed Atomicity for Flexible Transactions in Multidatabase Systems", *Proc ACM SIGMOD Conf*, pp. 67-78.

## **Terms and Definitions**

**Replication method:** A method for managing redundant data in such a way that a system can be optimized in some way.

**n-safe designs:** Replication methods where all n copies are consistent and up-to-date.

**1-safe designs:** Replication methods where 1 primary copy is consistent and up-to-date.

**0-safe designs:** Replication methods where none of the copies are consistent and up-to-date.

**Mirroring:** Replication methods where all updates to a logical disk volume are copied by the operating system to two or more physical disk volumes.

**Caching:** Replication methods where access to frequently used data is optimized. In *remote caching*, a primary copy of the frequently used data is normally stored on a very fast medium to optimize access to data. In *local caching*, an often inconsistent secondary copy of the frequently used data is stored in or close to the location of some users to optimize their access to the data.

**Availability of Data:** The probability of having access to the data. Replication will normally increase data availability.

**Read Performance:** The maximum number of read operations per time unit. Replication will normally increase read performance.

**Write Performance:** The maximum number of write operations per time unit. Replication will normally decrease write performance and give rise to a possible of inconsistency between the replicated data.

**Failure recovery:** Recovery in a situation where log-files and the current database or an old database copy are available.

**Disaster recovery:** Recovery in a situation where both the current database and its log-files have been destroyed.