Preface

This diploma thesis marks the end of my study in Computer Science at the Norwegian Institute of Technology. The work on the thesis was carried out at Telenor Research and Development's department in Trondheim.

I first started working with 1-Safe problematics in January 1995. For my spring term project [Tre95a] I was given the task of studying effects of applying a 1-Safe execution scheme for the ClustRa prototype. The work was continued during the summer months when Telenor employed me to implement a 1-Safe solution for ClustRa. This work is further developed with this thesis.

Chapter 1 and 2 gives an overview over the problem and presents the state of the art in 1-safe replication. The ClustRa prototype is presented in chapter 3. Behavior of a 1-Safe strategy in a system like ClustRa is given in chapter 4. Chapter 5 propose algorithms for coping with failures and recovering from these. The implementation and measurements are given in chapter 6.

I would like to thank all researchers at the department in Trondheim. They have been very helpful in answering my questions and provided a very inspiring environment to work in. I would especially like my two advisors Øystein Torbjørnsen and Svein-Olaf Hvasshovd for all their help both with technical problems and this paper. I have also received numerous comments and suggestions on the paper by Nina Trettenes and Håvard Filtvedt. Thanks a lot.

Trondheim, December 21. 1995

Tore Trettenes

Summary

This thesis describes how a 1-Safe execution scheme can be applied to a DBMS like the ClustRa prototype.

The ClustRa prototype is a DBMS providing very high service availability together with real time response and high throughput. In order to provide service availability across disasters like earthquakes, fires and landslides, data must be stored at two geographically distant locations. With todays 2-Safe execution scheme, updates must be reflected at both locations before the transaction is allowed to commit. This does not pair very well with the goal of real time response. The transmission delay to other sites will become the major factor in the response time for real time transactions running 2-Safe.

A 1-Safe execution scheme on the other hand allows replication of updates to be done outside the time critical path of a transaction. This means that the response time of a transaction running 1-Safe is independent of the distance to other sites. In contrast to a 2-Safe scheme, the 1-Safe scheme runs the risk of losing transaction when the system is affected by serious failures like disasters. This is the price for doing replication after transactions commit.

During the work with this thesis a 1-Safe execution scheme for ClustRa that employs parallelism both in log transferring and in the log installation at backup sites has been developed. Parallelism is a key issue to allow a system scale linearly with the number of nodes. Algorithms for dealing with failures and recovering from these are also presented.

The new 1-Safe execution scheme has been implemented and measurements show that the 1-Safe scheme outperforms todays 2-Safe execution scheme with respect to update transactions.

Contents

Preface i

Summary iii

Contents v

List of Figures ix

1 Introduction 1

2 State of the Art 3

2.1 1-Safe Replication with Multiple Parallel Log Streams 3

2.1.1 Dependency Reconstruction Algorithm 4

2.1.2 Epoch Algorithms 5

2.2 Parallel Log Installation 7

2.3 Commercial 1-Safe Products 8

2.3.1 Tandems RDF 8

2.3.2 Sybase Replication Server 9

2.3.3 INFORMIX-OnLine Dynamic Server 10

3 ClustRa Overview 11

3.1 Hardware and Database Model 11

3.1.1 Hardware Architecture 11

3.1.2 Database Model 12

3.2 Services and Transaction Management 12

3.3 High Availability 13

3.3.1 Replication 13

3.3.2 Node Supervisor 14
3.3.3 Takeover and Takeback 14
3.3.4 Self Repair 15
3.4 Logging 15
3.4.1 Distributed Log 15
3.4.2 Node-Internal Log 15
3.4.3 Log Installation 15
3.5 Recovery 16
3.5.1 Takeover 16
3.5.2 Catch-up and Takeback 17
3.6 Performance Measurements 17

4 1-Safe Behavior 19

4.1 1-Safe Overview 19
4.1.1 1-Safe versus 2-Safe Execution 19
4.1.2 Availability with Increasing Level of Replication 21
4.1.3 Surviving multiple node failures 22
4.1.4 Partially Controlled Takeover 23
4.1.5 Distributed Transactions 24
4.2 Losing Contact with Sites 26
4.2.1 Automatic Takeover Caused by Disaster 26
4.2.2 Takeover when Information is Insufficient 27
4.3 Assuring Consistency after Takeover 29
4.3.1 Rules for Installation of Updates 29
4.3.2 Identifying Lost Transactions 29

5 Recovery Using 1-Safe 31

5.1 Takeover Recovery 31
5.1.1 Ordered Shared Locks 31
5.1.2 Consistency with Exclusive Locking 32
5.1.3 Consistency with Ordered Shared Locks 34
5.1.4 The Effect of not Sending Read Operations 35
5.1.5 Making the Log Streams Consistent 37

5.2 Recovering Failed Replicas . . . 40
5.2.1 Very Serious Failures . . . 40
5.2.2 Main Memory Erased . . . 41
5.2.3 Main Memory Data Survived . . . 41
5.2.4 Recovery with Global Compensation . . . 42
5.2.5 Recovery with Local Compensation . . . 45
5.2.6 Scan Recovery . . . 47

6 Implementing 1-Safe in ClustRa 51

6.1 1-Safe Solution Properties . . . 51
6.2 Replication with Autonomous Databases . . . 52
6.3 Replication Inside ClustRa . . . 53
6.3.1 The OneSafeChannel . . . 54
6.3.2 The OneSafe Message . . . 55
6.3.3 Transaction Execution . . . 58
6.4 Measurements . . . 59
6.4.1 Test Conditions . . . 59
6.4.2 1-Safe versus 2-Safe with No Message Delay . . . 60
6.4.3 1-Safe versus 2-Safe with Far Site Replication . . . 61
6.4.4 Build-On Solution . . . 64
6.4.5 Conclusion . . . 66

7 Conclusion 67

A Measurements Data 69

B Source Code 75

Bibliography 107

List of Figures

2.1 The Epoch Algorithm 6

2.2 Replication in RDF 8

3.1 Hardware model for the ClustRa prototype 12

3.2 Execution of a simple 2-Safe transaction. 13

3.3 Multi-site mirrored declustering. -p are primary fragment replicas, -hs is hot standby. 14

3.4 The phases involved in recovery of a fragment replica at a crashed node. 16

3.5 The abort of transactions at a node. 17

4.1 Execution of a TPC-B-like transaction with 2-Safe replication. 20

4.2 Difference in replication costs for 1-Safe and 2-Safe. 20

4.3 Local node mirroring. 22

4.4 Local 2-Safe replication and 1-Safe inter-site replication. 24

4.5 A distributed transaction accessing data at two sites. 25

4.6 Is losing contact caused by site failure or network failure?. 26

4.7 The different failure modes when contact is lost. 27

4.8 The state machine of a site with events and actions. 28

5.1 Using exclusive locking at the backup to guaranty consistency. 33

5.2 Using Ordered Shared Locks at the backup to guaranty consistency. 35

5.3 No read operations available at the backup. 36

5.4 Compensation at the new primary replica. 38

5.5 Recovery with global compensation. 43

5.6 Recovery with local compensation 45

5.7 Different states of a record. 48

6.1 Replication laid on top of ClustRa. 52
6.2 Executing a simple transaction using 1-Safe replication. 54
6.3 OneSafeChannel state machine. 55
6.4 Layout of OneSafeMessage and TransMessage. 58
6.5 Throughput running 4-tuple transactions and no delay. 61
6.6 Response time running 1-tuple transactions and no delay. 62
6.7 Response time running 4-tuple transactions and no delay. 62
6.8 Response time running 1-tuple transactions and no delay. 63
6.9 Response time running 4-tuple transactions and no delay. 63
6.10 Response time running 1-tuple transactions and 5 ms delay. 64
6.11 Response time running 4-tuple transactions and 5 ms delay. 65
6.12 Response time running 1-tuple transactions and 5 ms delay. 65
6.13 Response time running 4-tuple transactions and 5 ms delay. 66

Chapter 1

Introduction

The high service availability demanded by some database users can only be met by also guaranteeing service availability after a disaster has occurred. In [Gra92] it is stated that there seems to be only one clear solution to this problem : applications should run on system pairs instead of just process pairs. This way applications can continue to run even if a total site should fail due to a serious disaster.

The idea is to keep the database distributed over geographically remote sites. This is done by letting all fragments having two or more replicas. Transactions are only allowed to access primary copies, and updates are replicated to secondary or hot stand-by copies. When applications in addition to these availability requirements also require real time response, the replication of updates will become the major component in transaction response time.

The demand for service availability will typically be 24 hours a day and 365 days a year with a maximum unavailability of a few minutes a year. This strict constraint would cause difficulties when hardware or software upgrades are to be applied to a centralized database system. With system pairs this is easily avoided. The upgrades are first applied to the backup site. Then the backup takes over as the primary site and the upgrades can be applied to the old primary site.

Several configuration options of a system pair is presented in [Gra92]. The different execution schemes differ in what actions will be taken upon commit-time. The configurations are called 1-Safe, 2-Safe and Very-Safe:

- **1-Safe:** With this strategy a transaction will commit at the primary and the log is transferred to the backup to be installed. The response time will be as low as with a centralized system. Therefore this is the strategy to choose if you require a system with extremely low response constraints and high throughput. The drawback is that when the primary is lost, the log with committed transactions may not have reached the backup and some of the committed transactions will be lost.

- **2-Safe:** A 2-safe strategy guarantees no lost transactions when the backup is up because a transaction will not commit before the updates are installed at the backup site. This delaying will increase response time and contention which will reduce throughput. If the backup is down, this strategy will run as the 1-safe strategy.

- **Very-Safe:** The very-safe strategy will not commit a transaction unless the backup

site is up and able to install the updates. This strategy is very strict and will block all processing of transaction if one of the systems become unavailable. Therefore this strategy is not used in practice.

To be able to scale up to databases with a large number of transactions per second, multiple parallel log streams will have to be used because a single log stream will quickly become a serious performance bottleneck. Even though the independent log streams arrives at the backup in strict order, it is not as simple as just installing the log as it is received. There must be some sort of synchronization between the log streams in order to maintain the backup consistency.

Chapter 2

State of the Art

Disaster recovery is a relatively new field in database technology. Nevertheless, much of the fundamental theoretical work has already been presented. Especially the team Garcia-Molina/Polyzois [GarPol94, GarPol90, GPH90, GPKH91] has contributed with a lot of work, but of course other researchers have also contributed [Lyo88, MoTrOb93, GPKH90, BurTre90]. Much of the presented work is further developed and republished in later articles, so I will therefore mainly pick from the latest publications where the ideas are most developed.

2.1 1-Safe Replication with Multiple Parallel Log Streams

The articles presented by Garcia-Molina, Polyzois, et al. address problems in disaster recovery and present algorithms using a 1-safe strategy to solve these problems. They have also implemented and evaluated the algorithms against 2-safe strategies.

The system architecture they consider is two sites with multiple communication-lines in between. Each site consists of multiple nodes, multiple stores, and multiple users can connect to each node. They expect multiple parallel log-streams from the primary to the backup. This is a requirement to allow the system to scale up. The transactions are assumed to be managed through a strict 2-phase locking concurrency control scheme.

In [GPKH91] the database consistency requirements for 1-safe transactions are presented. The three first are mandatory for algorithms with 1-safe update and the last one is to minimize the number of lost transaction in case the primary site is lost :

- **Atomicity.** If an action of a transaction T_i appears at the backup, then all write action of T_i must appear at the backup.
- **Mutual Consistency.** If some transactions have some dependency between them when they execute at the primary, they must have the same dependency ordering when installing their updates at the backup. Otherwise the database will become inconsistent.
- **Local Consistency.** A transaction with all its updates installed at the backup should not have dependencies to any lost transactions. This means that a transaction cannot

commit before all transactions it has dependencies to also have committed at the backup.

- **Minimum Divergence.** If a transaction is not missing at the backup and does not depend on any lost transactions, then its updates should be installed at the backup.

The algorithms they have presented are designed according to these consistency requirements.
They state that simply installing the log as it is received at the backup will violate consistency. There has to be some sort of local two-phase commit protocol to maintain consistency. Therefore the algorithms consider how to install updates at the backup such that the backup remains consistent with the primary.

Garcia-Molina and Polyzois have also presented some performance tests for the algorithms described above. In general these tests show that the Epoch Algorithms and the Dependency Reconstruction Algorithm almost always outperforms 2-safe strategies. More details of the performance tests can be found in [GarPol94].

2.1.1 Dependency Reconstruction Algorithm

The Dependency Reconstruction Algorithm is presented in [GPKH90]. It guarantees consistency at the backup by ordering the transactions at commit-time and commit the transaction in the same order at the backup. Upon commit-time at the primary a transaction receive a ticket from the nodes where it accessed data. Each node keep a counter to order the transactions. There is one ticket-process at each node to avoid these processes from becoming global bottle-necks. This puts restrictions on the location of hot stand-by fragments. All primary fragments must have their corresponding hot stand-by fragments on the same backup node. This node replication means that if a primary node should fail, the corresponding backup node must be able to process all transaction requests intended for the failed node.

One of the primary nodes will act as coordinator of a transaction. The coordinator will make a lists of accessed nodes and sends this list to the coordinating node at the backup. All the nodes will be coordinators for multiple transactions.
At the backup site, the coordinating node will check that all updates of a transaction are installed by checking with the accessed nodes. When it has received acknowledgment from all accessed nodes, it sends the commit message to the accessed nodes and also the coordinating node at the primary.

If a transaction has written any records at a node it will get a ticket, say Counter(24) + 1, and the ticket-counter is incremented to Counter(25). If the transaction only read records, it will also get a ticket Counter(24) + 1, but the counter will not be incremented. This means that RW-dependencies will not be considered at the backup.

When log streams arrive at the backup the transactions will pass through four states before their changes are made permanent in the database :

- **LOCKING :** The transaction T_i arrives at the backup requesting locks for all the records it wants to update. It will not enter the SUBSCRIBED state before all transactions with lower ticket numbers have entered or gone past the SUBSCRIBED state.

This restriction allows all transaction that can possibly be in conflict with T_i to request the locks they want and thus avoid all conflicts when installing updates.

- **SUBSCRIBED :** The transaction waits until it is granted all the locks it requested. Then it enters PREPARED state. Conflicts are avoided because locks are granted in ticket order.

- **PREPARED :** An acknowledgment for the first phase of the two-phase commit protocol is sent to the backup coordinating node.

- **COMMITTED :** The coordinating node have received acknowledgment from all accessed nodes. The message for the second phase has been sent to the accessed nodes. All changes have been made public and all locks are released.

The proof for the correctness of the Dependency Reconstruction Algorithm is given in [GPKH91].

2.1.2 Epoch Algorithms

The Epoch Algorithms takes a different approach than the dependency reconstruction algorithm. Instead of synchronizing updates at the backup, an epoch algorithm is just more careful when it sends the parallel log streams to the backup. The log is sent to the backup in packages called epochs. The nodes at the backup can install updates independently when all nodes have received its next epoch. The log is divided into epochs at the primary in such a way that a transaction have the same state at all nodes when the epoch is ended: committed, not committed, or aborted.
The algorithm assumes node replication just like the Dependency Reconstruction Algorithm in the previous section.

Single Mark Algorithm

One of the nodes at the primary is master for ending epochs. All nodes have their own epoch-counter. When the master decides to end an epoch, it increments the local epoch-counter, writes an end-epoch record to the log and send an end-epoch message to the other nodes. When the nodes receive the end-epoch message they also increment their local epoch-counter and writes an end-epoch record to the log.

To make sure that transactions have the same state when an epoch is ended, the local epoch number is included when the coordinator of a transaction sends out the pre-commit message. The nodes receiving this pre-commit message compares the number in the message with its local epoch counter: If the local epoch counter is less then the number in the message, the node knows that a new epoch is started. Therefore the node will increment its local epoch counter and write an end-epoch record to the log before writing prepare-record to the log and sending ready message to the coordinator.
If the received epoch number is less than the local epoch number, the ready message with the local epoch number is sent and will force the transaction coordinator to change epoch before committing the transaction.

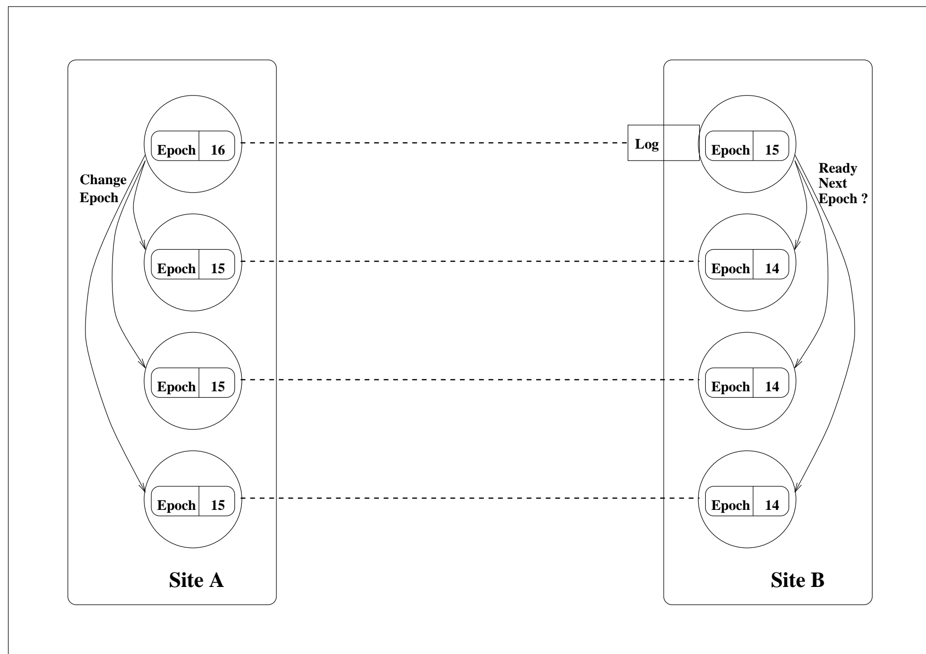

Figure 2.1: The Epoch Algorithm

At the backup the log installation can begin when all the nodes have seen the end of the next epoch to be installed. There are a set of rules the nodes can use to decide the state of a transaction:

- If the commit record is in the epoch, the transaction is committed at that node. The rest of the nodes will reach the same decision.
- If the prepare record is in the epoch, the node can ask the transaction coordinator for a decision. If the node is the transaction coordinator, the decision will be not to commit the transaction during the current epoch.
- If none of the above is true for a transaction, it cannot be committed during the current epoch.

The correctness of the algorithm is showed in [GPH90].

Double Mark Algorithm

This algorithm is a variation of the previous. The difference is that it does not have to include epoch numbers in messages back and forth from the transaction coordinator. This makes it easier to use in existing systems, because no modifications to the internal message formats are necessary.

Instead of including epoch numbers in messages this algorithm has two rounds of messages to end an epoch. First the master sends a pre-end-epoch message. When the nodes receives

this pre-end-epoch message they stop making any new commit decisions, write pre-end-epoch records in their log and send acknowledgment to the master node. When the nodes receive the second end-epoch message, they write the final end-epoch entry in the log and resume normal operation.

It is worth noting that even though the nodes must delay the commit of transactions to after the last end-epoch message, they can still process requests in between the two end-epoch rounds.
This algorithm also has rules to decide what state a transaction is in at the backup. These rules can be found in [GPH90].

2.2 Parallel Log Installation

In [MoTrOb93] an algorithm for maintaining a hot stand-by site is presented. Like all previous algorithms it relies on shipping the log to the backup where it is installed to make the backup consistent with the primary. The unit of update is a database page.

The algorithm apply parallel log installation at the backup, but all log have to pass a single control point. The received log records are hashed into a great number of work queues. Conflicting log records will have the same ordering after being hashed into work queues as in the log received log at the backup. The paper suggests one work queue per database page. In practice this was too expensive, so each work queue is set to administrate the updates to a number of pages. When installing the updates, one redo process for each work queue can run in parallel.

2.3 Commercial 1-Safe Products

A number of commercial systems have been presented to provide systems with higher availability and reliability. I will now give a brief presentation of some of these systems. They all use the shared-nothing hardware model which provide a high degree of failure isolation with respect to software failures and environmental failures such as fires, floods, land slides and earthquakes. The other thing they all have in common, is the use of a single log stream from the primary site to the backup site. This can become a performance bottleneck as the number of nodes at sites increases.

2.3.1 Tandems RDF

Tandems Remote Duplicate Database Facility (RDF) [Gue91, Lyo90]has been around for some years. Tandem have earlier developed fault-tolerant systems by duplicating hardware and software at a centralized site. RDF can be viewed as a natural extension to these systems to also provide a system that can cope with disasters.

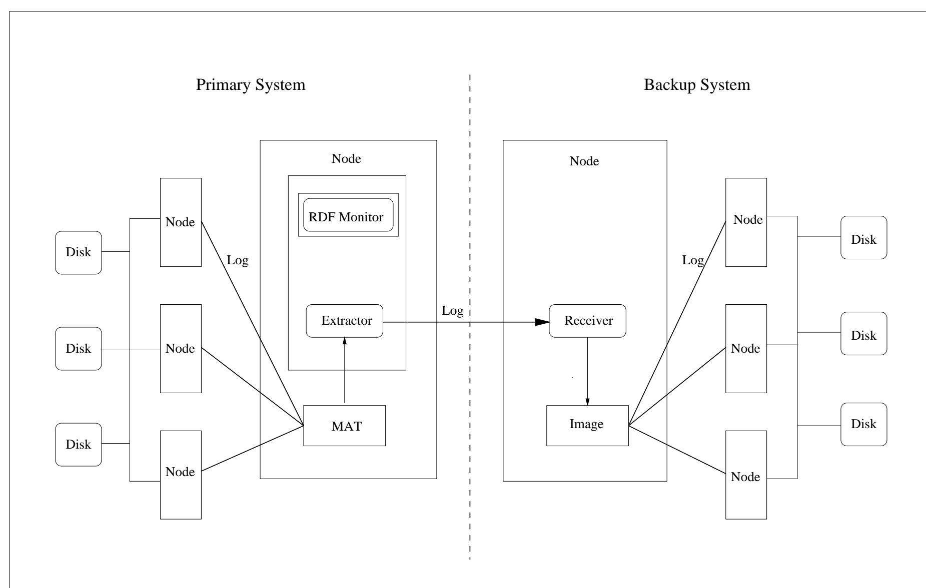

Figure 2.2: Replication in RDF

RDF is built on top of Tandems Transaction Monitoring Facility (TMF) which is a part of Tandems operating system and provides transaction atomicity and recovery. Since RDF only depends on TMF, all applications that use TMF can also use RDF without any modifications. TMF maintains the log of the database transactions in a Master Audit Trail (MAT) which is used by RDF.

In a RDF system the different sites are given different roles to play: one acts as primary system and the other as backup system. At the primary system an RDF monitor is set to scan the Master Audit Trail. If the monitor sees a log record that should be reflected at the backup site, the log record is copied by an extractor process and sent over to the backup site

where it is saved in an image file by the receiver process. This is illustrated in Figure 2.2. The log records of a transaction are applied to the database when the commit record for the transaction is received.

Takeover processing is achieved in two different ways: controlled or through site failure.

The roles of primary and backup can be switched during normal operation. The primary is asked to switch role. It stops accepting new transaction, processes the transactions in transit and takes the role as backup site. Before it switches to the backup role, it writes a switch-record in the log. When the backup sees this record, it takes the role as primary site and starts accepting transaction requests. This controlled switch-over can be used when hardware or software changes are to be applied to the system without decreasing user availability.

The second way of takeover is when the primary site fails. Then the backup system is manually told to take over the role as primary site. The backup system processes all outstanding log records. If the commit message of a certain transaction is not received at the time disaster struck, the updates are bypassed. When all log records are applied or bypassed, the database will be in a logically consistent state, but some transaction committed at the primary may be lost.

A problem with RDF is that it only supports one log-stream. This can make it difficult to scale up to support databases with a large number of transactions per second.

2.3.2 Sybase Replication Server

SYBASEs Replication Server [Syb92] is a system with an open architecture. This means that a distributed database system can be built from existing systems and applications. It can also support heterogeneous data servers.
Each table have one primary copy and can have multiple replicated copies. Servers with replicated data can subscribe to the primary tables they want to have up to date.
The system consists of three components:

- **Data Server:** This can be an already existing system. The server provides data availability to end users.

- **Log Transfer Manager:** The Log Transfer Manager reads the log of a data server. All updates done to primary copies of data, will be sent to the Replication Server.

- **Replication Server:** The Replication Server sends updates to the Data Servers subscribing for the updated data.

When building a system with existing data servers, these must supply their own log transfer manager if the data server is not a SYBASE product.

2.3.3 INFORMIX-OnLine Dynamic Server

The INFORMIX-OnLine Dynamic Server [Inf94] provides both 1-Safe and 2-Safe replication consistency policies. High availability is provided by detection and masking of DBMS site crashes by automatic client switch-over. The unit of replication is all data at a site. Only primary or hot stand-by replicas is allowed at a single site. The internal DBMS internal log i used as the basis for replication, where each log record contains a complete before- and after-image of the updated tuple.

This system maintains an I-am-Alive protocol between the primary and the backup site. INFORMIX-OnLine Dynamic Server can perform automatic takeover, but this is done only based on missing I-am-Alive messages. The unit of failure is a whole site when using the 1-Safe policy. When a network partitioning is treated as a site failure, this system can get serious inconsistencies in the database.

Chapter 3

ClustRa Overview

ClustRa [HvToBrHo95] is a Database Management System (DBMS) currently being developed by Telenor Research. It is supposed to provide low response-time, high throughput, and high availability. The motivation for developing a new database system is that none of the existing systems can provide both the performance and the high availability necessary for telecom applications. When starting the development of the prototype, certain goals was established:

1. **Throughput:** At least 1000 TPC-B-like[1] or lighter transactions per second.
2. **Response time:** Maximum 15 milliseconds response time for at least 95% of the TPC-B-like transactions. Aborted transactions are included in the transactions not meeting the response time.
3. **Availability:** No more than 1 hour unavailability over 30 years mission time. The mission time is 24 hours a day, 365 days a year. The availability measuring interval is 60 seconds.

These requirements seem very strict, but anything less will not be adequate for telecom applications. The first two goals have been reached by the project [HvToBrHo95]. Only the last and most difficult remains. In the following sections I will give a brief description of how the different parts of the system works today.

3.1 Hardware and Database Model

3.1.1 Hardware Architecture

The ClustRa prototype uses a shared-nothing hardware model. Each site consists of multiple nodes which have their own disks and memory. Nodes communicate with each other through an ATM switch. An Ethernet can also be used for communication. Figure 3.1 shows the hardware model.
Because the nodes at a site do not share anything, they will have almost independent failure

[1] Benchmark described in [Gra91]

modes to both software and hardware failures. However, when facing a disaster they will have common failure modes given their collocation. Therefore the database will be replicated at a remote site for protection against disasters.

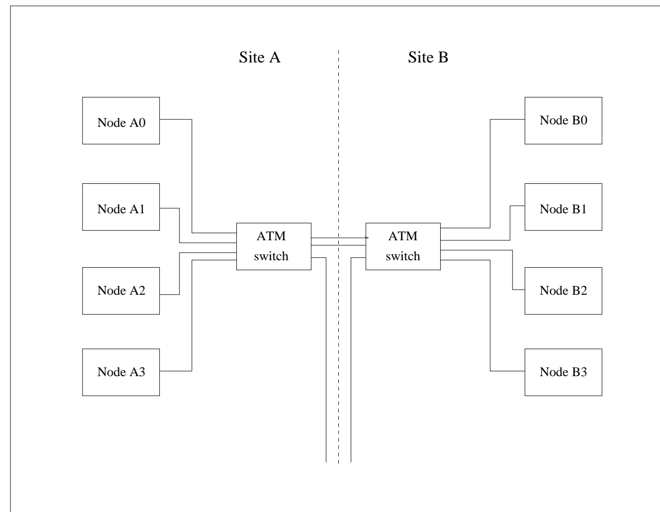

Figure 3.1: Hardware model for the ClustRa prototype

A node-failure will be much more common than a disaster. The architecture minimizes the chance for multiple simultaneously node-failures in which case a disaster may have to be declared. In case of node-failure, the corresponding hot-standby-node will take over processing. At the primary site a cold spare will be replicated and take over as the new primary.

3.1.2 Database Model

ClustRa employs a relational database model. The tables are fragmented horizontally based on primary record keys either through a hash function or a key range. Each node is set to process requests to records stored in its own memory. Fragments are the unit of replication. Each fragment have one primary fragment and one or more hot stand-by replicas allocated at different nodes.
Both the fragmentation and physical location of fragments may be different at the primary and hot stand-by site. Therefore fragments at a primary node can be distributed among several nodes at the hot stand-by site. This will result in a more even workload at the hot stand-by site in case of a primary node-failure.

3.2 Services and Transaction Management

A set of main processes are responsible for managing the user transactions and guaranteeing a serializable execution of the transactions:

- **Kernel(KERN):** The Kernel is responsible for local data management at a node. All read and update requests from TCON processes are executed by the KERN process.

- **Update Channel(UCHN):** The Update Channel is responsible for sending update records to hot stand-by replicas.
- **Transaction Controller(TCON):** The Transaction Controller is responsible for executing user transactions. There is one TCON process for each transaction. This process may send requests to one or more KERN processes. The TCON can also communicate with a TCON process at the remote site which takes care of committing the transaction at the backup if the primary should fail.
- **Node Supervisor(NSUP):** The Node Supervisor is responsible for collecting information about service availability and informing about changes in service availability.

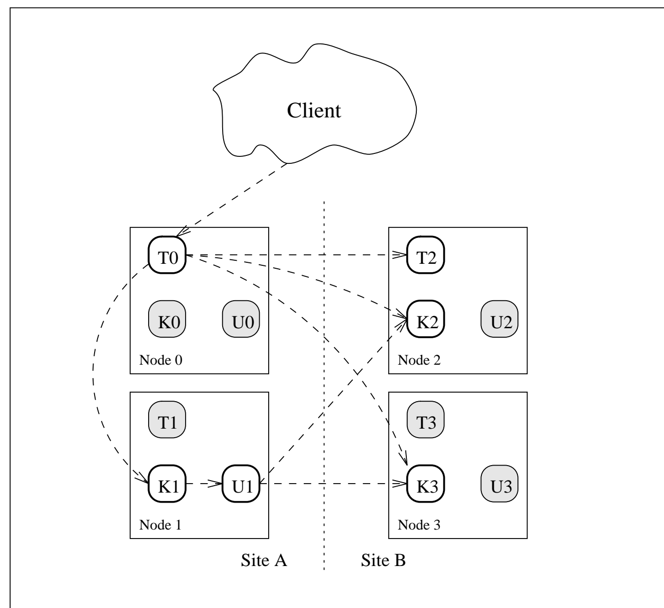

Figure 3.2: Execution of a simple 2-Safe transaction.

The main processes responsible for executing user transactions and their interaction are illustrated in Figure 3.2.

3.3 High Availability

3.3.1 Replication

High availability is ensured through data replication. For each data fragment there will be maintained multiple replicas, which are assigned to be a primary or hot stand-by replica. Replicas of the same fragment is placed at nodes with different failure modes. The number of replicas per fragment can easily be defined in data distribution files read at start-up time. The system will need at least two replicas per fragment placed at geographically separate locations to provide availability both after single node failures and after disasters where a whole site fails.

Figure 3.3 shows an example the multi-site declustering strategy. There is one primary and one hot stand-by replica placed at different sites. It is also possible with other replica placement strategies where primary replicas at one node have their corresponding hot stand-by replicas at different nodes at the backup site. This provides better load balancing after a node failure than the node-to-node replication in Figure 3.3

Site s_0				
Node	Fragment replica			
n_0^0	0-p	4-hs	8-p	12-hs
n_0^1	1-p	5-hs	9-p	13-hs
n_0^2	2-p	6-hs	10-p	14-hs
n_0^3	3-p	7-hs	11-p	15-hs

Site s_1				
Node	Fragment replica			
n_1^0	0-hs	4-p	8-hs	12-p
n_1^1	1-hs	5-p	9-hs	13-p
n_1^2	2-hs	6-p	10-hs	14-p
n_1^3	3-hs	7-p	11-hs	15-p

Figure 3.3: Multi-site mirrored declustering. -p are primary fragment replicas, -hs is hot standby.

Replication alone is not enough to assure very high availability. First of all there must be some way of detecting a failure. Depending on the type of failure, certain actions must be executed. After a failure some of the replicas in the system will be unavailable and the system is not providing the promised level of protection against additional failures. Therefore it is important to return to the normal level of protection as soon as possible.

3.3.2 Node Supervisor

Each node have a node supervisor responsible for checking the status of the processes running within the node. It is also responsible for sending I-am-alive messages to its neighbor nodes. The messages are sent with a high frequency to detect changes in the system as soon as possible. If some messages are missing, the I-am-alive protocol is activated. The unresponsive node is given a second chance to say that it is alive. If the node is still not responding, it is assumed that the node has failed and the protocol announce this fact to all nodes in the system. Upon receiving such a message, a node will know whether it should take over for some fragments or not based on information from the data distribution files.

3.3.3 Takeover and Takeback

When a node fails, some primary replicas will become unavailable. To provide users with access to the whole database, one of the hot stand-by replicas must become the new primary replica for the failed fragment. This takeover procedure will result in one less hot stand-by replica ready to take over in case of additional failures. It is clear that the system is vulnerable to additional failures as long as the system runs with fewer replicas for some fragments than it would under normal operation. Therefore the system will try to restart the failed node as soon as possible and recover the failed replicas. If the node can be restarted, the replicas can be recovered with help from the current primary replicas. When a failed primary replica are recovered, it can announce its recovery and take back the responsibility as primary replica. Takeover and takeback is presented more thoroughly later in this chapter.

3.3.4 Self Repair

If it is impossible to do a fast recovery of the failed replicas, the system will start self repair. This means that new replicas are made as compensation for the failed replicas. This will allow the system to return to the fault-tolerance level it had under normal operation. The generation of new replicas is done on-line based on data from the current primary replica of a fragment.

3.4 Logging

ClustRa uses two types of logging. The distributed log is logical and contains the logical operations done to tuples in the database. This log is used to keep hot stand-by replicas up to date and to provide transaction durability. There is also a node-internal logging for representing changes in the physical structures such as block management and changes to the B-tree structures.

3.4.1 Distributed Log

In the ClustRa prototype, fragments are the unit of replication and logical log streams go between primary and hot stand-by fragments. There are many parallel log streams from primary to backup, but they are multiplexed and share the same communication lines. In each log stream the log records include a fragment identifier to ensure consistency at the backup.

The distributed log is used for logical redo and undo and replication. As operations are executed by a kernel, records with both before and after image of the updated tuples are put into the distributed log. This log is read by the update channel which send update log records to the hot stand-by replicas maintaining a copy of the updated tuple.

3.4.2 Node-Internal Log

The Node-Internal log reflects internal operations such as access methods, free block management and file directory operations. This log is disk-based and is not shipped to any other node. A failed node can be restored to a consistent state by redoing the node-internal log first and redoing the logical log afterwards. The two different logs together avoids forced disk flushes in the time-critical transaction execution.

3.4.3 Log Installation

In the previous section the UCHN process was described. This process extracts update records from the primary log and sends them to the corresponding hot stand-by replicas. At the backup they are installed in the same order as they are received.

Before a transaction commits at the primary, the TCON process has made sure that the updates are installed at the backup site. It has also told a hot stand-by TCON process at

backup site that it is committing the transaction. In case the primary TCON should fail, the hot stand-by TCON will know the status of all transactions ready to commit.

3.5 Recovery

When a failure is detected in the system, certain actions are being executed. The responsibilities of a failed primary replica will be taken over by one of the hot stand-by replicas. The system will also try to recover from the failure and return to normal operation. The different phases from a node failure back to normal operation is shown in Figure 3.4.

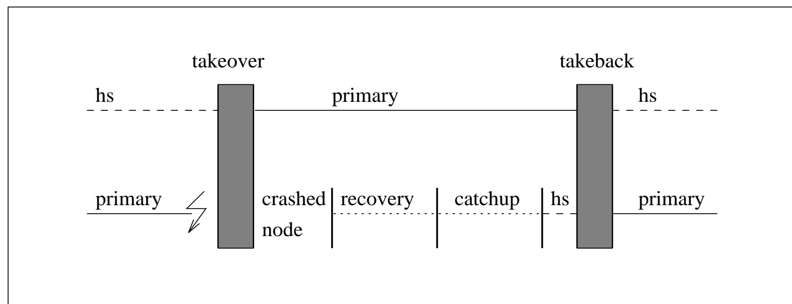

Figure 3.4: The phases involved in recovery of a fragment replica at a crashed node.

3.5.1 Takeover

The node supervisor will detect and inform all nodes in the system if a node should fail. In the data distribution files there is information about which of the hot stand-by replicas should take over as the new primary replica. This means that when a node receives information about a failed node, it will know whether it should take over for some replica. It will also know for which active transactions it can start producing CLRs for. Note that CLRs can only be produced by the primary replica.

Replicas taking over as new primaries, must be made consistent with respect to uncommitted transactions. The decision whether to commit or abort a transaction is made by the new primary TCON process:

- If the TCON (hot stand-by TCON at the time) was informed that the transaction was ready to commit and sent acknowledge for this, then the decision will be to commit the transaction. When a transaction is ready to commit all its slaves is also ready and updates will be installed at two sites.
- If no such message was received before the failure, the decision will be to abort the transaction. Messages from ready slaves will be responded with an abort message.

Note that if the slaves were not ready at the time of the node failure, they can make a local abort decision because all slaves must be ready to commit a transaction. An example of aborting a slave at a node is shown in Figure 3.5

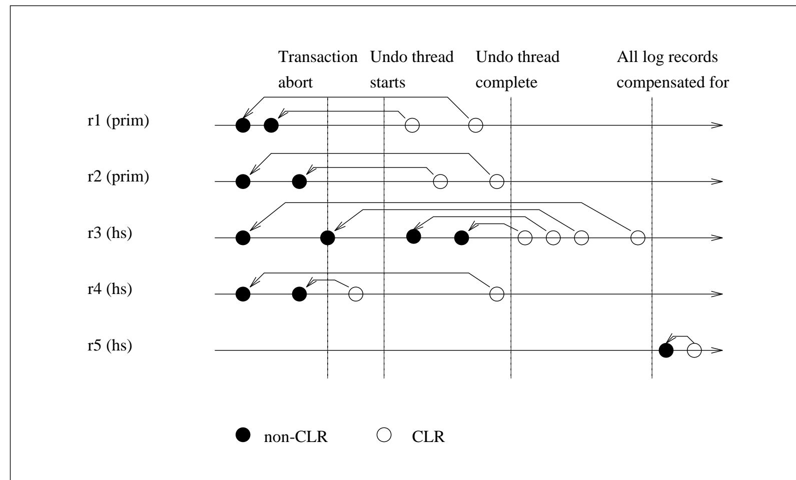

Figure 3.5: The abort of transactions at a node.

3.5.2 Catch-up and Takeback

In order to return to normal operation, the system must try to recover the failed replicas or produce new replicas. As long as the level of replication is reduced, the system is vulnerable to additional failures.

A failed node can often be restarted right away without having lost any fragment data or log. Figure 3.4 shows the phases of a recovering replica. The first phase consists of recovering to the last stable state. Checksums are used to see if data in shared memory survived the failure. If data in shared memory is corrupted, recovery from disk will be started.

When the internal recovery is finished, the recovering replica announces that it is recovering. The current primary replica responds by sending compensation log records produced at takeover time and catch-up log reflecting updates done after takeover. The first record received by the recovering replica will be the bump-up record which tells which log record was last received. The recovering replica will then produce compensation log records for all log not received by the current primary replica before the failure. When this is done, compensation log and catch-up log from the current primary will be applied. The recovering replica will then become a hot stand-by replica.

After having become a hot stand-buy replica, the recovering replica can takeback the responsibilities as primary by announcing that it is about to become primary again. The current primary replica will stop accepting new requests and become a hot stand-by replica. This whole process brings the system back to the normal data distribution described in the distribution files read at startup.

3.6 Performance Measurements

The ClustRa prototype have reached its goals with respect to transaction response time and throughput. In [HvTo95] measurements which show this are presented. A 16 node

configuration ran a TPC-B-like benchmark 2-Safe on replicated data giving a throughput of 1052 transactions per second with an average response time of 11,3 milliseconds. These benchmark tests were run without physical distance between the logical sites. With a longer distance between the sites, fewer transaction will be able to finish within the 15 milliseconds.

Chapter 4

1-Safe Behavior

4.1 1-Safe Overview

This section will give an overview of characteristics of the 1-Safe execution strategy. To see how these characteristics affect the system, they are in some of the sections compared with the corresponding characteristics of a 2-Safe execution strategy.

4.1.1 1-Safe versus 2-Safe Execution

The end users will get faster replies on their database requests with a 1-Safe execution strategy. This holds for all requests needing replication to other sites. The reason for this can be found in the nature of 1-Safe and 2-Safe strategies:

- With a 2-Safe strategy, the end user will not get a reply before updates done by the transaction are received at another site. The system may also wait until all replication is finished before giving the end user a reply.
- With a 1-Safe strategy, the system will not wait for any far site replication to be finished before giving a reply to the user.

The clear advantage of the 1-Safe execution strategy for the users, is the reason why more and more high availability systems provide this mode of operation. High availability systems must replicate updates to other sites with independent failure modes towards disasters.

When a system also wants to provide real time response for its users, the difference between 1-Safe and 2-Safe becomes even greater.

Another reason for using a 1-Safe replication strategy, is the ability to lower the costs of replication by reducing the number of messages generated and thereby reduce the amount of communication between sites. I have tried to quantify the difference in replication costs under the 1-Safe and 2-Safe strategies. An execution scenario is showed in Figure 4.1.

In this scenario, 4 replicas is maintained per data fragment. Two replicas are placed at two different sites. The TPC-B-like transaction updates four tuples at two different local primary fragment replicas. The scenario corresponds to some of the measurements done for

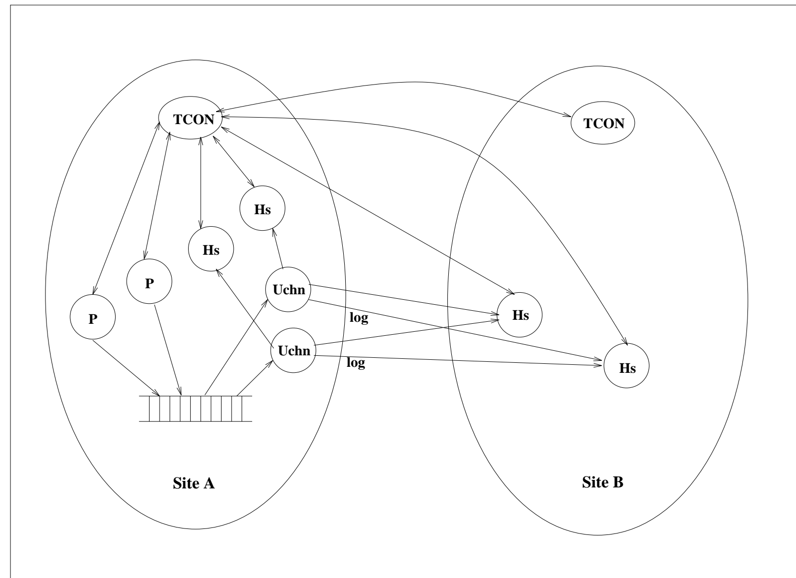

Figure 4.1: Execution of a TPC-B-like transaction with 2-Safe replication.

ClustRa. Depending on replication strategy the following number of inter-site messages will be generated in ClustRa:

- **2-Safe:** Every update generate a log message being sent to two different nodes at the other site. All these messages will be replied with an acknowledgment message. This adds up to 16 messages for replication purposes. The 2-phase commit process will generate another 12 messages.

- **1-Safe:** Log messages will be grouped just as commit messages will. This means that the number of messages per transaction will be amortized as many transactions are grouped.

If we assume that the 1-Safe strategy groups about 50 transactions before sending log and commit messages, the 1-Safe strategy will generate 12 inter-site messages. It should be noted that the 1-Safe strategy will generate more messages within each site. The total number of messages generated for executing 50 transactions with 1-Safe and 2-Safe:

Messages Generated for 50 Transactions		
Type	1-Safe	2-Safe
inter-site	1400	12
local	700	1200
total	2100	1212

Figure 4.2: Difference in replication costs for 1-Safe and 2-Safe.

4.1.2 Availability with Increasing Level of Replication

As mentioned earlier the reason for maintaining multiple replicas of a fragment is to increase the data availability as experienced by end users and to reduce the probability of losing all replicas of a fragment. Losing all replicas of a fragment can be seen as a very serious data unavailability and break on reliability.

It is clear that data availability increases as the number of replicas maintained for each fragment is increased, but it will of course be more expensive to maintain increasing number of replicas. If a system start maintaining twice as many replicas per fragment, the cumulative probability of a node failure will be twice as high, and the system will use twice as much resources on recovering and repairing failed replicas.

I will use level of replication to designate the number of replicas per fragment. The 1-Safe and 2-Safe replication strategies both have advantages and disadvantages which depends on the level of replication. Below I will describe the properties of 1-Safe and 2-Safe replication as the level of replication is increased from 2 to 4 replicas per fragment. An interesting fact is that 1-Safe will become more similar to 2-Safe as the level of replication is increased. The reason for this is that when employing a 1-Safe strategy the local replication can be done using 2-Safe replication without receiving a large penalty in response time.

My study is concerned with how a system may survive a site failure, the lowest number of replicas allowed is two. These two replicas must be placed at geographically different sites. A third replica is placed at the site where the primary replica is residing and the fourth replica is placed at the other site.

- **2 Replicas:**
  - **2-Safe :** Node failure results in a node takeover, a disaster will result in a site takeover with no lost transactions.
  - **1-Safe :** Node failure will result in a site takeover which may cause lost transactions.
- **3 Replicas:**
  - **2-Safe :**Node failure will result in a local node takeover, a disaster will result in a site takeover with no lost transactions.
  - **1-Safe :**Node failure will result in local node takeover with no lost transactions, a disaster will result in a site takeover with transactions lost.
- **4 Replicas:**
  - **2-Safe :**Same as for 3 Replicas, but the system can now run with only one site operational and still survive a single node failure.
  - **1-Safe :**Same as for 3 Replicas, but the system can now run with only one site operational and still survive a single node failure.

This comparison shows what will happen if either a single node fails or if the complete site fails. All nodes are expected to have the same probability of failure and they are also expected to be failure-independent with respect to failed nodes.

In general a 1-Safe replication scheme can provide the same availability by maintaining one more replica than 2-Safe. It is also possible to achieve the same availability with no extra replication. To accomplish this, the logical log must be sent to a second local node before a transaction commit. Should the primary replica fail, the logical log at the second node will be sent to the hot stand-by replica as catch-up log before it becomes primary.

The probability for multiple nodes to fail simultaneously is very low compared to the probability of single node failures, but should nevertheless be considered. In general the 2-Safe protocol will have advantages over a 1-Safe protocol, as a multiple node failure may cause a site takeover when using 1-Safe replication. It is clear that it is very important with 1-Safe replication to have access to at least one replica of all fragments at a site. If all replicas of one fragment become unavailable, the whole site must accept a site takeover because of the inconsistency introduced by the unavailable fragment. The inconsistency arise from the fact that some transactions will be lost and these transactions may also have done updates to other primary replicas. If these primary fragments continue to accept operations they will be violating the ACID-rules as they may reflect updates done by a lost transaction.

4.1.3 Surviving multiple node failures

For the 1-Safe strategy it is important that a site survive a multiple node failure thus avoiding a site takeover. If the system maintains N replicas at a site, the site will survive as long as no more than N-1 nodes fails simultaneously. But since many replicas per fragment will introduce extra costs, it is more interesting to look at what chances a site have for surviving with as few replicas as possible, which for the 1-Safe strategy is two. In Figure 4.3 a scenario with 8 nodes at a site having two replicas of each fragment is illustrated.

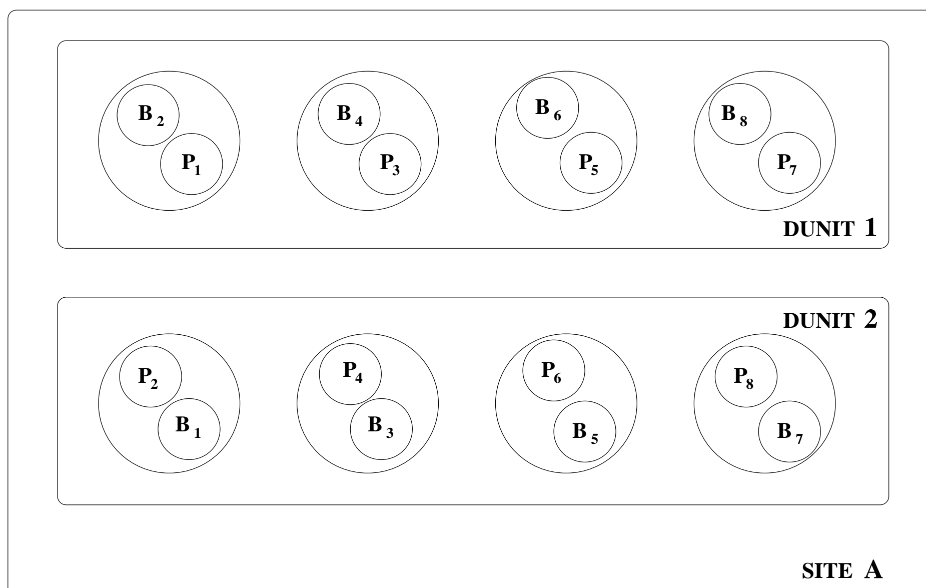

Figure 4.3: Local node mirroring.

As the figure shows, two and two nodes replicate each other. This placement of the replicas leads to the highest probability for surviving a double node failure. With a 1-Safe strategy

a site takeover will only be necessary when the two nodes holding replicas for each other fails simultaneously. With this configuration the chance for a site to survive a double node failure is 85%. For configurations with 4 and 16 nodes the corresponding probability of surviving a double node failure is 66% and 93% respectively. This probability equals inverse the number of remaining nodes. Note that the cumulative probability for a node failure grows as more nodes are added to a configuration.

The probability for multiple nodes to fail simultaneously is very low, but when a node have failed there will be a bigger chance for a second node failure before the first node have totally recovered. To avoid this second node failure from making some fragment unavailable and thus initiating a site takeover, the system can employ self-repair. This means that when a node fails, leaving just one replica for some fragment, the system will start producing a second replica for this fragment without waiting for the failed node to recover. If the replica is produced before the second node failure, the site will be able to provide access to all fragments. The use of self-repair will increase the probabilities given in the previous paragraph. There will always be a question whether the remaining nodes at a site will be able to provide full service with the loss of CPU resources introduced by the multiple node failures.

4.1.4 Partially Controlled Takeover

Above I have only considered the problems of data availability. If there is insufficient processing power to give full service to users, the site or nodes themselves can decide if they should let another node takeover for a while. This controlled takeover is much less serious than a failure as losing transactions can be avoided and thus avoid the introduction of inconsistency in the database. This controlled takeover (or "giveover") can also be done on a per fragment basis. When the site is ready to become primary again, the concerned fragment can be re-possessed in a takeback (or "giveback").

The ability to execute a giveover and giveback procedure on a per fragment basis can be used for load balancing both during normal operation and during failure. Consider the configuration presented in Figure 4.3. If a double node failure should make all replicas of some fragment unavailable, the correct action will be to do a site takeover. This site takeover can then be partly controlled as most of the nodes are still operational. To execute a controlled takeover the remaining nodes must stop excepting new transaction, wait for ongoing transactions to finish or abort them and wait until the backup site has received all log produced by all its fragments. Then these nodes are ready to become hot stand-by for all its fragments. At the backup site, the new primary replica will be restored to the last consistent state. Note that if all log is received by a hot stand-by replica becoming primary, this fragment will not introduce any lost transactions to the system. This should make it easier to restore the fragment to its last consistent state. The last consistent state will now be completely decided by the failed fragments and their lost transactions.

When the new primary replica have been restored, it can be given back on a per fragment basis assuming the owner is not one of the failed nodes. It is worth noting that in a distributed system the primary replicas are placed according to access patterns of end users. Therefore a strategy of giving back single fragments or even single nodes, will probably introduce a higher ratio of distributed transactions.

4.1.5 Distributed Transactions

The term transaction used earlier in this report, has been used about transactions that only execute operations against data that is primary at the local site. Distributed transactions on the other hand will execute operation against primary data residing in multiple sites. With 2-Safe replication these transactions will not introduce any particular problems other than extra round trips between sites resulting in higher transaction latency.

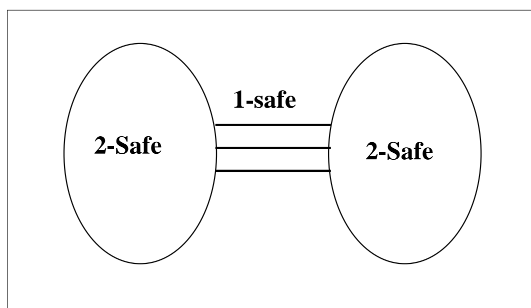

Figure 4.4: Local 2-Safe replication and 1-Safe inter-site replication.

Figure 4.4 shows a configuration where the local replication is done 2-Safe and replication across different sites is done using 1-Safe replication. This means that all primary replicas and its corresponding hot stand-by replicas at a site will be consistent with one another. A site will also hold hot stand-by replicas of fragments which have primary replicas at other sites. These replicas will be maintained using 1-Safe replication and will not be consistent with the rest of replicas at the site, but after a site takeover they can be restored to a state that is consistent with the other replicas and thus be a part of a consistent database. How a replica should be maintained can be stated as follows:

> *All primary replicas and hot stand-by replicas that could become primaries must be kept consistent by using 2-Safe replication, or have the ability to be restored to a consistent state.*

With only local transactions it becomes clear that after a takeover, no lost transaction will have executed operation to any of the primary replicas at the site taking over and thus these fragments will not need to take part in restoring the inconsistent fragments to a consistent state.

This is where distributed transactions turns out to be a problem. Distributed transaction will access primary replicas at different sites as shown in the Figure 4.5. The user transaction executes updates to both P_1 and P_2. Site 1 is the site receiving a request from the client. This means that B_1 will be kept consistent with P_1, but B_2 is maintained through 1-Safe replication and will not be consistent with P_1or B_1. If site A should fail, replica B_2 will become the new primary. Before B_2 can become a primary replica it must undo updates done by lost transaction. Lets say the distributed transaction is lost and thus committed, but not all updates reached B_2 from P_2 before the failure occurred. Now B_2 must undo the updates done by the distributed transaction, but the same transaction also updated P_1. These updates should also be undone, but it will be expensive to stop all execution to a

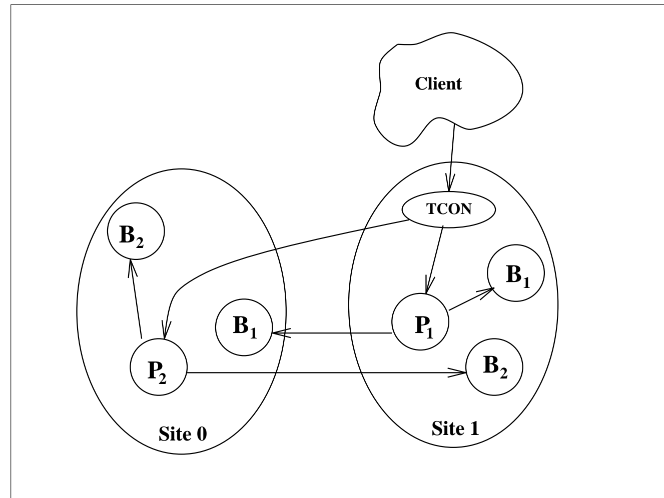

Figure 4.5: A distributed transaction accessing data at two sites.

primary replica, because this would lead to unavailability. A way to avoid this problem is to force distributed transactions to do all replication using 2-Safe replication. Distributed transaction do have a blocking effect because they hold locks longer than 1-Safe transactions, but the ratio of distributed transactions should be low in a well distributed system.

Other transactions may also run 2-Safe. This could for instance be transactions that would be expensive to lose, e.g. bank transactions with big deposits or redrawals.

4.2 Losing Contact with Sites

When a site loses the contact with another site, it will not know whether this is because the communication network failed or if the other site actually was affected by some sort of disaster. In the following, a site will be a candidate for a site takeover if it is not possible for other sites to communicate with any of the nodes at the affected site. If any nodes at the site affected by some sort of failure is still alive, these nodes can estimate the damage done and if necessary ask other sites to take over its responsibilities. If none of the nodes survived the failure, the site is said to be a failed site.

4.2.1 Automatic Takeover Caused by Disaster

The problem with distinguishing between a failed network and a failed site is well known and can be illustrated as in Figure 4.6.

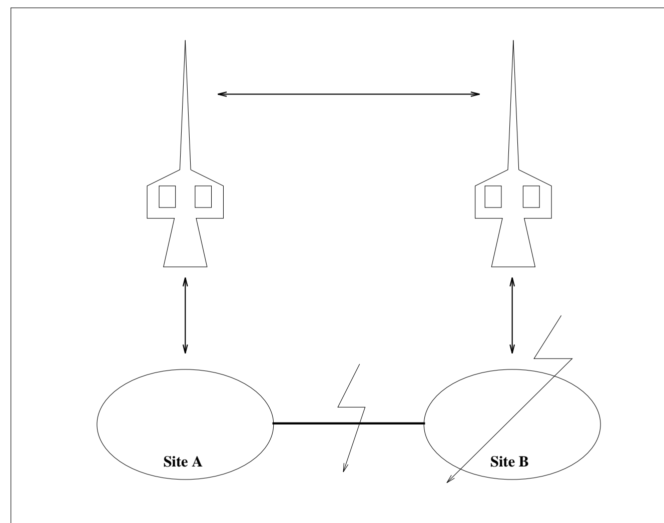

Figure 4.6: Is losing contact caused by site failure or network failure?.

Site A does not know if the communication lines were broken or if Site B failed. This is the reason why commercial products offering protection against disasters will need manual intervention to conduct a site takeover. The time needed for a database administrator to discover and perform the takeover, will be unacceptable to systems with very strict availability requirements, e.g. telecommunication applications. To be able to decide the type of failure, the system can use a model where sites are connected with different sets of communication lines which are fairly independent with respect to failure. This is indicated in Figure 4.6 with radio links providing alternative communication lines between Site A and B. The radio links can also be affected by disasters like an earthquake. Therefore the model does not guaranty a correct decision, but it increases the probability of a correct decision. The alternative communication could also be handled by satellite links, but for most purposes this solution seems much to expensive.

The alternative communication lines can help Site A to decide whether Site B has failed

or not. If Site B did not fail, the alternative communication lines could also be used for ordinary replication, provided the capacity is sufficient.

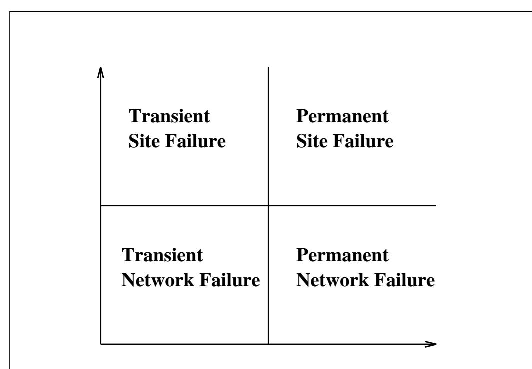

Figure 4.7: The different failure modes when contact is lost.

When Site A knows with high probability what kind of failure caused the loss of contact with site B, it can act according to this information. But since the 1-Safe replication scheme makes the takeover procedure more expensive than the corresponding 2-Safe procedure due to lost transactions, it would be wise to give the failed site a chance to recover before a takeover is conducted. With two types of failures and different durations of these failures, we can put system failures into four different categories as indicated in the illustration below. Transient failures indicate failures where the site or network is able to recover within a certain time frame. If it is not possible to recover within this time limit, the failure is said to be permanent and proper actions will be executed.

In Figure 4.8 a state machine for a site is shown. This state machine tells how a site should act when it loses contact with other sites. It contains the different states a system can be in, the corresponding actions to be executed and the events causing state transitions.

4.2.2 Takeover when Information is Insufficient

If alternative communication lines is to expensive or not available, it is not possible for a site to determine what type of failure caused the loss of contact. This means that some design decisions have to made about what actions to execute when contact with the other sites is lost. There seems to be a lot of options and choices to deal with this "what happened"-problem.

At the time the system is affected by some failure, certain actions will have to be executed to secure consistency and data availability. It is assumed that a site through the use of local replication can recover from a single node failure and therefore there will not be necessary to conduct a takeover.

There are several different actions to choose when contact between sites is lost:

- Mark one of the sites as primary for all data when contact is lost. This will make the marked site a single point of failure which breaks with the shared-nothing architecture.

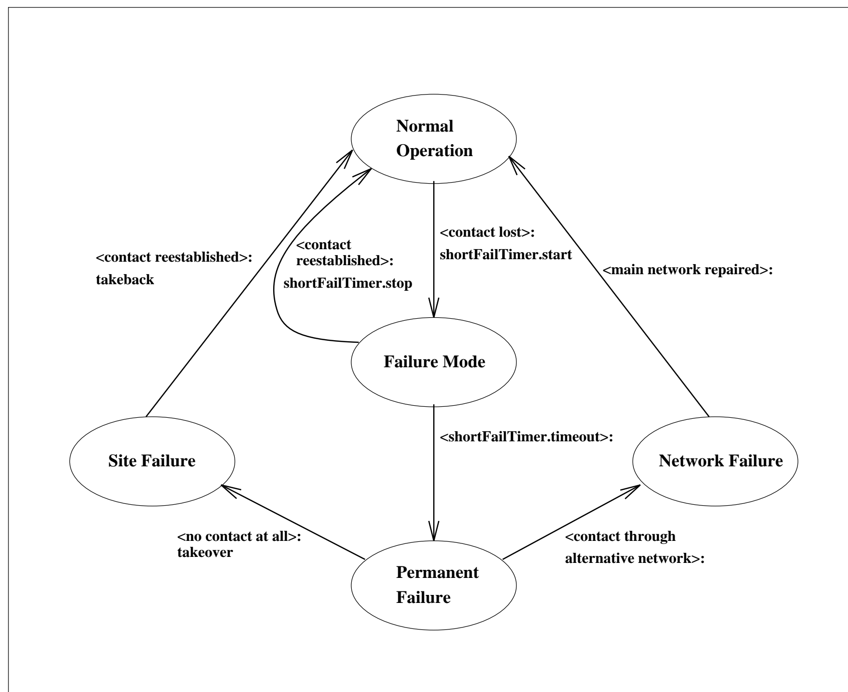

Figure 4.8: The state machine of a site with events and actions.

If the marked site fails, all data will become unavailable. If users connected to a failed site is not able to switch to the primary site, they will experience unavailability.

- If communication is controlled by intelligent switches, the can tell the system if communication lines were broken. But these switches might also fail and not being able to tell if communication lines are broken.

- Treat loss of contact as site failure. This means that network failures will be treated as site failures. There will become multiple primary replicas for each data fragment, which may cause persistent inconsistencies.

- Treat loss of contact as network failure. This means that site failures will be treated as network failure and fragments being primary at the failed site will become unavailable. If the system is well distributed, a network failure will not be very serious as most transaction only access local data.

Assuming that single node failures can be masked within each site, the probability of a network partitioning is higher than the probability of a site failure. This leads to a combined strategy where network partitioning is assumed first and together with some additional criteria site failure is assumed:

4.3 Assuring Consistency after Takeover

4.3.1 Rules for Installation of Updates

When it has been decided that a site should take over as primary for another site, certain requirements will have to be met to ensure that the database consistency is not violated when hot stand-by replicas become primaries. These requirements was presented in Section 2.1. Locks is used to describe dependencies between different transactions.

I will use operations instead of locks to describe dependencies. This means that the requirements will look a bit different because a write-lock can also be used for reading, and thus creating more WR-dependencies. These WR-dependencies may be non-existent as not all write-locks will be used for reading. This is why I chose to distinguish between locks set at the primary and the operations executed:

1. **Atomicity.** If a write-operation of a transaction **Ti** appears at the backup, then all write-operations of **Ti** must appear at the backup.
2. **Mutual Consistency.** If some transactions have some dependency between them when executing at the primary, they must have the same dependency ordering when installing their updates at the backup. Otherwise the database will become inconsistent.
3. **Local Consistency.** A transaction with all its updates installed at the backup should not have WR-dependencies to any lost transactions. This means that a transaction can not commit at the backup before all transaction it has WR-dependencies to also have committed. It is neither allowed to commit before transactions it has WW- or RW-dependencies to have either committed or aborted.
4. **Minimum Divergence.** If a transaction is not missing at the backup and does not have WR-dependency to any lost transaction, then its updates should be installed.

These requirements will guaranty consistency for the hot stand-by replicas before they take over as primary fragments. The system will use a 1-Safe protocol described in Section 1 for replication between sites. We will have to make some rules for this protocol that contain the requirements described above.

4.3.2 Identifying Lost Transactions

After a site failure, a transaction can either be installed or aborted depending on the information received at the backup site before the site failure occurred. A Transaction that was not committed at the primary site before the failure, will be aborted at the backup site by restoring the before-images of all tuples updated by the transaction. Transaction that committed at the primary site may or may not be able to get installed at the backup site. If the transaction cannot be installed without violating the database consistency it is regarded as a lost transaction. To identify the lost transactions we must look at what information made it to the backup site before takeover and dependencies between lost transactions and transactions not yet installed. If log and commit-messages are sent independently, there are four different situations for a transaction:

1. **all log received and commit received:** The transaction can be installed provided that there are no dependencies to any lost transactions.
2. **all log received and commit not received:** Abort the transaction.
3. **all log not received and commit received:** Abort the transaction.
4. **all log not received and commit not received:** same as above.

A transaction that have received the commit-message at the backup, but not all its log records will be aborted and be regarded as lost.

Chapter 5

Recovery Using 1-Safe

5.1 Takeover Recovery

I have developed three different algorithms for returning to normal operation after a site takeover. The algorithms are also valid for single node failures if local replication is employed. If there is no local replication, a single node failure will cause a site takeover.

The first algorithm uses exclusive locking (strict 2PL) at the backup as used in most locking-based DBMS to guaranty the requirements described in the previous chapter. The second algorithm will use Ordered Shared Locks for the same purpose. Most readers will probably be familiar with 2PL, but since fewer will know of Ordered Shared Locks, I will give a short presentation in the next section.

5.1.1 Ordered Shared Locks

The parallel log-streams received at the backup sites have some nice properties that can be exploited to make log installation simpler.

Log records received at the backup site will never cause dead-locks. The log records are redone in the order they are received in the log-streams. Presuming the transaction commits at the primary site, the corresponding "redo"-transaction at the backup will never have to abort if it receives all log-records.

Transactions with dependencies between them have to commit in the same order at the backup site as they did at the primary site. This is ensured by locking the log records as they are redone and unlocking all updated records when the transaction is ready to commit. This means that a transaction with a dependency to some other transaction will be blocked when trying to update an already locked record.

In [Agr95] a new locking policy is presented to avoid the blocking nature of the 2PL policy. The motivation for this is that blocking transactions in a real time system will degrade performance. The presented policy allows different transactions to obtain locks for the same record. This is called Ordered Shared Locks. Transactions getting ordered shared locks must execute according to the Ordered Sharing Rule:

If $\mathbf{T}_j$ *acquires a lock with an ordered shared relationship with respect to a lock held*

by another transaction $\mathbf{T}_i$*, the corresponding operation of* $\mathbf{T}_j$ *must be executed after that of* $\mathbf{T}_i$*. Furthermore,* $\mathbf{T}_j$ *cannot commit until* $\mathbf{T}_i$ *terminates (commits or aborts).*

5.1.2 Consistency with Exclusive Locking

Exclusive locking is used by many DBMS as concurrency control to guaranty that execution comply with the ACID-rules. When log records are received at the backup site for installation, there will also have to be some sort of coordination in the installation process. The reason for this is the use of multiple asynchronously log streams. With only a single log stream from the primary to the backup site, a single controller can take care of the the necessary coordination to ensure the log installation rules described in Section 4.3.1. With multiple log streams there will also be multiple controllers which do not know how the installation is doing at other nodes. To be able to allow nodes install updates independently, conflicting updates must be detected and installed according to the log installation rules. This can be solved by using exclusive locking.

If the log records get locks as they are received at the backup, any conflicting updates received at the backup will be blocked until the locks are released. Remember that a transaction may have updates received by multiple nodes at the backup, and a single node can not decide when all log records for some transaction have been received. No lock will therefore be released before the atomicity rule is fulfilled for the transaction being installed.

A locking scenario after a site failure is shown in Figure 5.1. In the figure $\mathbf{t}_{90}$ to $\mathbf{t}_{97}$ are tuples and the circles symbolizes operations waiting to be installed. New log records are received at the left, so the leftmost log records are the ones received last. The index of an operation denotes the transaction owning the operation. The scenario may seem a bit constructed and in a real situation it is unlikely that a similar scenario will arise. The scenario is used to illustrate all possible situation and what actions must be executed in each situation.

The figure shows the last log records received at the backup before the primary site failed. The circles with dashed lines are log records that did not get to the backup before the site failure. They are just meant to show which transactions that can not be installed without violating the atomicity rule.

If we use the log installation rules in Section 4.3.1, the following will happen:

1. Transaction number 1, 9 and 10 can not be installed because they are missing updates.
2. Transaction number 3 and 4 can not be installed because they read values from transaction number 1.
3. Transaction number 5, 6 ,7 and 8 can be installed when the lost transactions have been compensated for.

As mentioned above, locking is used to guaranty that the requirements for installing updates are not violated. These requirement must be met both under normal operation and after a site takeover. Single node failures is not seen by any of the nodes at the backup site because another node at the primary site is expected to take over all the responsibilities of the failed

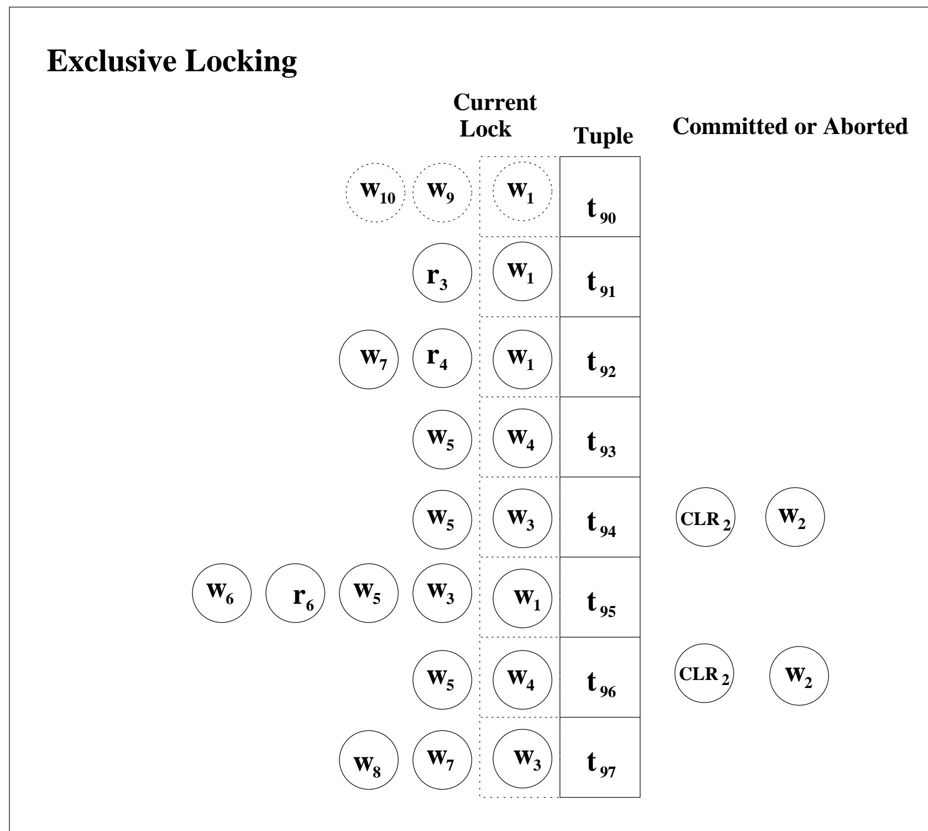

Figure 5.1: Using exclusive locking at the backup to guaranty consistency.

node. Multiple node failures are also possible to mask within a site, but may also force the site to ask another site to take over.

When a site fails, the logical log streams will be cut. Some streams may have lost many log records while other streams may have delivered all operations produced at the primary. When a transaction have committed at the primary, it can be given a deadline for when it must be installed at the backup. This time limit will be big enough for allowing all transaction to be installed during normal operation of the system, but after a site failure transaction will reach this deadline if they can not be installed because of missing records. The question that will arise when a transaction can not be installed, is what other transactions should be aborted:

- If updates is done to the whole tuple, transactions that have directly read values written by a lost transaction must be aborted. This means that the next write operation on some tuple will stop the search for transactions to abort.
- If updates is done only to some attributes of a tuple, the aborting transaction must abort all subsequent transactions that read any attributes of the tuple. This is necessary to make sure that no transaction is installed when they have read attributes written by a lost transaction.

An abort-procedure must in addition to restoring before-images, also check for these WR-

dependencies to other transactions. If some transaction read a value written by a lost transaction it cannot be installed without violating the consistency of the database and must therefore also abort and look for transaction with WR-dependencies to itself. This makes the abort-procedure cascading. The cascade stops when there is no more WR-dependencies to be found:

ABORT($\mathbf{T_i}$) :

> *Check the lock request queues for tuples written by* $\mathbf{T_i}$. *Send ABORT(*$\mathbf{T_j}$*) to all transactions* $\mathbf{T_j}$ *that read a value written by* $\mathbf{T_i}$ *at the primary if updates are done to the whole tuple. If updates are only done to some attributes of a tuple, send ABORT(*$\mathbf{T_j}$*) to all transactions with a read request for the tuple. When the search for transaction to abort is terminated, the before-image of tuples written by* $\mathbf{T_i}$ *must be restored.*

It should also be stated that the abort-procedure will only be used when it has been decided that the other site is down and no more log records will be received. In fact, the procedure does not allow any log records to be received after the first transaction is about to be aborted. This restriction is enforced because log records received after the takeover decision may have WR-dependencies to aborted transactions which will not be discovered.

5.1.3 Consistency with Ordered Shared Locks

Using Ordered Shared Locks at the backup to guaranty consistent installation means that multiple transactions can hold locks to the same tuple at the same time, but they will have to commit in the same order as they received the locks for the tuple. There may exist several uncommitted versions of a tuple at the same time. Identifying which transactions must be aborted after a site failure is not different from Exclusive Locking, but the transactions must restore all accessed tuples to the previous version.

A scenario where ordered shared locks is used is shown in Figure 5.2.

With exclusive locking this is not a problem as there will only be one uncommitted copy of a tuple created by the lost transaction. With Ordered Shared Locks the aborting must also tell subsequent transactions with dependencies to restore the previous copy of the tuple. These Compensating Log Records (CLR) must be produced in the opposite order from the operations executed on a tuple. After the committed version of the tuple have been restored and all lost transactions and transactions with dependencies to lost transactions have been aborted, the remaining transactions may redo their operations and commit as usual. The ABORT-procedure presented in the previous section will have to be adjusted to work with ordered shared locks. This procedure is both cascading and recursive:

ABORT($\mathbf{T_i}$) :

> *Check the dependency queues for tuples written by* $\mathbf{T_i}$. *Send ABORT(*$\mathbf{T_j}$*) to all transactions* $\mathbf{T_j}$ *that read or wrote a value written by* $\mathbf{T_i}$ *at the primary. Wait for acknowledgment that these transactions did abort. Put back before-images to written tuples.*

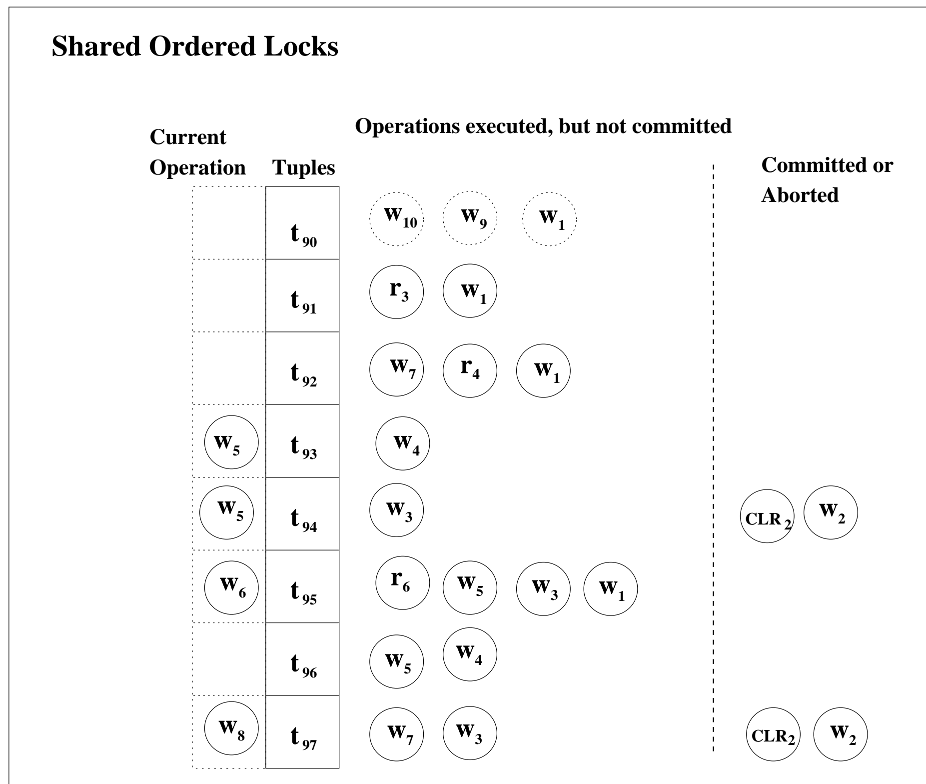

Figure 5.2: Using Ordered Shared Locks at the backup to guaranty consistency.

This procedure works either updates are done to the whole or only to parts of a tuple. Operations of transactions that are not regarded as lost, may redo their log records and commit when the abort-procedure have terminated.

5.1.4 The Effect of not Sending Read Operations

If a system could be allowed to not send read log records a lot of communication costs would be saved as the fraction of read operations is high for a typical telecom application. But what effect will this have on the consistency of the database? We can use the scenario from Section 5.1.2 and just leave out the read operations.

It is clear that under normal operation it is not necessary to have the read operations at the backup. This is because all updates will eventually reach the backup and the locking will guaranty that conflicting updates is installed according to the requirements described in Section 4.3.1. The effect of not having read operations at the backup is that transactions that would have been blocked because of some WR-dependency, no longer will be blocked and can commit earlier than allowed by strict consistency rules.

After a takeover, the installation may cause inconsistencies in the database because WR-dependencies is not discovered. This will allow transactions that should have been aborted because of WR-dependency to some lost transaction to be installed. The wrongful installation will be transactions that have read an uncommitted value and may have written

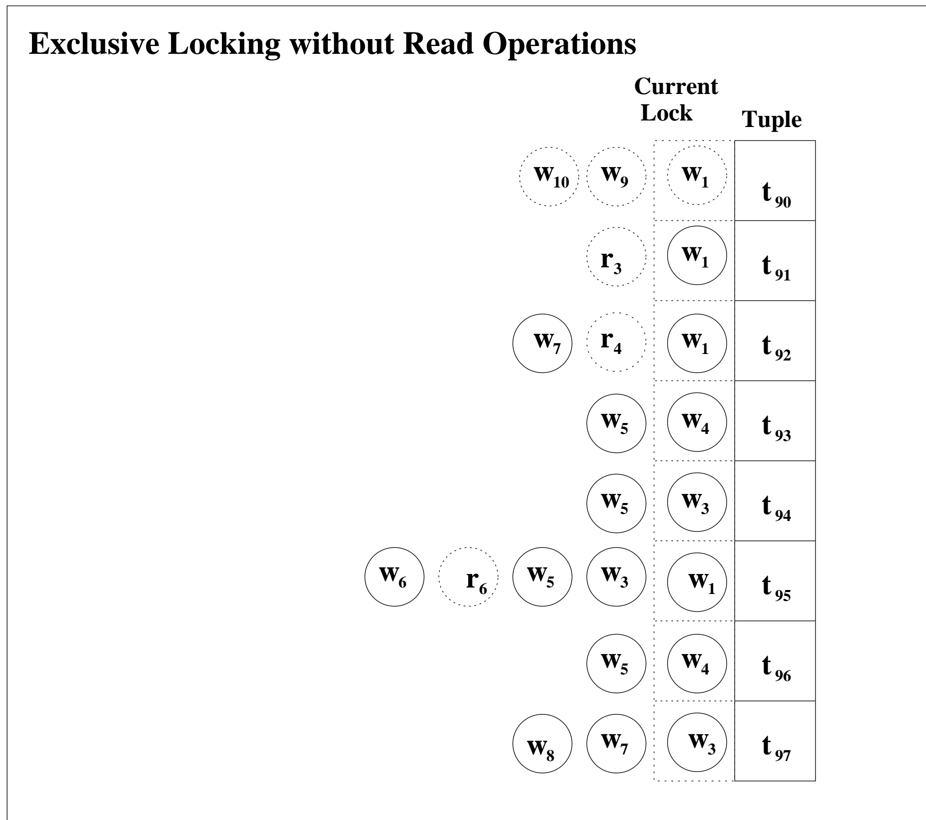

Figure 5.3: No read operations available at the backup.

tuples based on the read value. If the system continue to execute transactions against an inconsistent database, additional inconsistencies may be created. But it is also possible that inconsistencies may be disappear as inconsistent tuples can be written by transactions reading only consistent data.

If we look at the scenario in Figure 5.3, the following will happen:

1. Transaction number 1, 9 and 10 can not be installed because they are missing updates.
2. Transaction number 3 and 4 will be incorrectly installed because they have no read operation to block them. They will cause inconsistencies because they read values from transaction number 1.
3. Transaction number 5, 6 ,7 and 8 will be correctly installed when the lost transactions have been compensated for. In this particularly scenario the inconsistency created by transaction number 3 and 4 are wiped out by the updates of transaction number 5 and 7, but this is not the case in general.

Note that a 2-Safe replication strategy does not need read operations to appear at the backup to guaranty consistency at the backup. The reason for this is that transactions conflicting with these read operations will be blocked at the primary fragments due to the concurrency control employed there.

5.1.5 Making the Log Streams Consistent

The algorithms described in Section 5.1.2 and 5.1.3 guaranty the consistency of a fragment after a takeover. This consistency is assured at transaction level. But what about the log streams? The log streams should reflect the same updates that is reflected in the fragment. Remember that the log stream is used during recovery to bring a replica back to a consistent state.

With a 2-Safe replication scheme the log streams will always reflect the correct changes done to data in a fragment. The reason why this holds, is that a transaction will not commit at the primary before changes are reflected in both log streams and data at both primary and hot stand-by replicas. If a transaction is about to be aborted, the abort procedure will make the log streams consistent with respect to this transaction before it is aborted. This is done by producing Compensation Log Records (CLR) for all the state changing operations of the transaction to be aborted.

The situation is different when using a 1-Safe replication scheme. Transactions committed at the primary have no possibility to help the backup with synchronizing its log streams to reflect the right updates. This means that when a transaction cannot be installed after a takeover, the installation algorithms must also make sure that the log streams are consistent.

The locks at the backup will help the installation algorithm with keeping track of updates that are not installed. When a transaction cannot be installed, the abort procedure can just produce CLRs for updates that should not be reflected in the database. This may sound simple to do, but remember that the algorithms described involved a lot of analysis of which other transactions with WR-dependencies should be aborted. Looking for these dependencies may not be easy, as there may not be such a thing as a lock request queue for each tuple. Therefore a real system would look for a very simple solution that will also guaranty the consistency given in the algorithms above.

With locking at tuple level the minimum divergence requirement is guaranteed. No transaction will be blocked unless it has operations that conflicts with operations of some other transaction. This means that if we give all transaction enough time to be installed at the backup, the remaining transactions when this time frame has elapsed will be transactions with missing updates (lost transactions) or transactions with some dependency to lost transactions. The log streams can now be made consistent by producing CLRs for updates belonging to uncommitted transactions. A takeover scenario where a log stream is made consistent with the the fragment data is shown in Figure 5.4.

In the figure the BUMP-UP record is used to show where the logical log streams was broken at takeover time. The log stream is made consistent by scanning the log stream backwards and producing CLRs for updates that is not committed. In the figure the different log records are:

- W for state changing operations such as INSERT, UPDATE, DELETE and delta-operations(increment and decrement).
- C for signaling COMMIT of a transaction locally at a site.
- GC for signaling that a transaction is successfully replicated at another site. This records is called GLOBAL COMMIT and corresponds to the COMMIT operation when using 2-Safe replication.

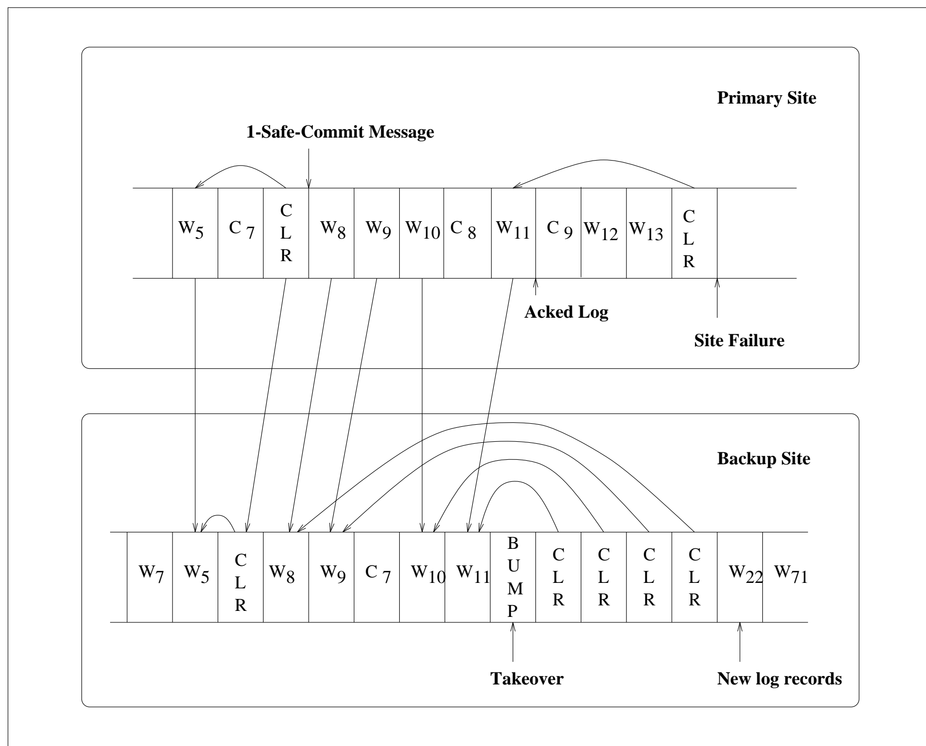

Figure 5.4: Compensation at the new primary replica.

- BUMP for signaling where the logical log stream was cut in a takeover.
- CLR for Compensation Log Record which is used to undo updates.
- A for signaling that a transaction is ABORTED both locally and globally.

Detailed description of the Log Synchronizing Takeover Algorithm :

1. The algorithm will only be started when it has been decided to conduct a takeover. First of all the algorithm will produce a BUMP-UP record to know the last log record received from the failed primary replica.
2. When all correctly received have been given enough time to get installed, the log stream will be scanned backwards to find uncommitted updates. The scan will have a list of uncommitted transactions when the scan starts:
   - For each W record a CLR which undoes this operation if a COMMIT record for this transaction have not been seen.
   - For each R record nothing is done, but the LSN is incremented
   - For each C record nothing is done but the corresponding transaction is registered as committed and should not have any of its updates undone. The LSN is also incremented.
   - For each A record nothing is done, but the LSN is incremented

- For each CLR record nothing is done, but the LSN of the compensated record is registered and the LSN is incremented.(The compensated record will be found later in the undo scan and can also be ignored. The alternative would be to compensate for both the compensation and its corresponding non-CLR.)

The undo scan continues until there are no transactions to undo. A transaction is aborted when the last update have been compensated for.

5.2 Recovering Failed Replicas

When the system experiences a permanent site failure, some other site or several sites in cooperation will take over as primary for the fragments that became unavailable. The new primary replicas will provide access to user transactions as long as the failed site is down. When the failed site has been repaired or restarted, the system must take back the failed site to return to normal operation again.

The failed site will have to synchronize its failed replicas with the replicas that took over as the new primary replicas before it can be allowed to become primary again. While the failed site is synchronizing its data copies, the site is said to be in "catch-up"-mode. The failed site are not allowed to accept any user transactions before it is fully synchronized and have data copies identical to the operational sites. When the sites are fully synchronized, the primary site is informed that the recovering site is ready to become primary for the fragments assigned to it in the fragmentation and replication files. The primary site will now stop accepting new transaction to data copies it became primary for after the site failure. These fragments will be "given" back to the recovering site.

This whole operation of recovering a failed site might not seem to hard too do, but there are many problems to be solved. Some of these arise from the nature of the 1-Safe protocol used for replication between sites where losing transactions are allowed. These transaction are committed at the primary site but did not make it to the other site before the site failed. Another problem is how much of the data was lost when the site failed. A serious failure means that the data in main memory was lost and perhaps disks were damaged too. If it is possible to reload some older data copies from disk, the recovering site may have been down so long that the site with the new primary replica was unable to keep all redo-log while the site was down.

It should be noted here that all failures causing a site takeover may be considered serious, especially when local replication of fragments lets a site recover from a single node failure without any help from other sites. Nevertheless, some failures are harder to recover from than others depending on how much information was lost in the failure.

5.2.1 Very Serious Failures

With a very serious site failure we can expect that all data in main memory was lost and that the disks were damaged. This means that new hardware would have to be bought and the whole database will have to be downloaded from other sites. Many of the lost transaction may never be identified or retrieved in any way.

If there is more than two sites in the system, the sites that did not fail can cooperate in reloading the recovering site and thus make the recovery faster. This method can also be used when the failed site loses some of its data by discarding what ever data would be left when recovering. However the method tends to be expensive since the whole database will have to be transferred across the communication network leading to large communication costs. The nodes at a failed site may very well have different levels of failures causing their takeover by some other node. This is because the nodes does not share any resources and therefore may have different failure modes even to disasters. Maybe some nodes only lost main memory data while others lost all data including disk resident data.

5.2.2 Main Memory Erased

We can have a situation where data residing in main memory is lost but where less up to date copies of the data can be retrieved from the disk system. In the ClustRa prototype, check-pointing guarantees that data copies loaded from disk are at most two checkpoints older than the data copy lost in the failure. The failure will erase both data and the corresponding log in main memory, so two checkpoints of log may be completely lost if it cannot be retrieved from some other site. Writing an update to disk is only allowed if the log record reflecting the update is already on disk. This means that the copy reloaded from disk will not contain any updates described in the two checkpoints of lost log. The log is in addition to being written to disk, also shipped to some peer node at another site as part of the replication process. This shipping is done as the log records are created and after a failure, the peer will have much more recent log than the two checkpoints old log available from disk. In addition all log that was not received at the backup will have to be redone when recovering the failed site to assure that the database consistency is not violated.

After the database is reloaded locally, the lost transactions can be undone and new changes to the fragments redone. The lost transactions can be written to a special log so that their updates can be merged when the system have returned to normal operation. It is unlikely that all lost transactions can be identified or at least retrieved, but most of lost transaction can be found in the log received at the backup site.

From the database theory we have that transactions can be executed in what sequence we want and still get equivalent histories if the operations of the transactions does not conflict with the operations in any other transaction. The merging will then just consist of redoing the log records of the lost transactions. If you on the other side get into conflicting operations, you will have to choose what is worst of losing the transaction or the possibly of violating the database consistency.

To make the recovery from disk a bit simpler, there can be put some restrictions on when the system is allowed to flush changes to disk. One restriction could be not to allow the log to be written to disk before all transaction have been successfully replicated at the other site. This means that when the main memory is erased, all lost transaction will be completely lost. When reloading from disk, all changes reflected in the data was successfully received at the other site and thus no undo is required. Of course, there may be transactions that is successfully replicated at the backup, but still not written to disk at failure time. These will be received from the backup site as a part of the catch-up log. The catching-up phase will now consist of redoing transactions that was not lost and transactions processed after takeover.

There may be a problem to identify the lost transactions based on the log sequence numbers. This also depends on what happens to lost transactions at the site taking over as primary. These problems will be described in the next section.

5.2.3 Main Memory Data Survived

In the ClustRa prototype, data and log is placed in shared data segment as described in Section 3.1.2. A situation where the update channel, transaction controller, and kernel processes dies with their internal data structures, but where the shared data segment survives the failure is likely.

When no data is lost in the site failure, the data copies can be recovered and reappear as primary copies when fully synchronized with the current primary copy. This process will be much faster than the recovery from disk. The recovery time is dependent on the time the site was down and the redo-ratio. The redo-ratio tells us how fast the redoing of log records can be done compared to the speed at which log records is produced at the primary replica.

We will have a transient failure when another site did not take over as primary before the failed site was ready to start the recovery process. This means that no new transaction was executed during the site failure and thus the failed site can be recovered by just restarting the main processes. No transactions will be lost provided the shared data segments survived the failure. This will typically be a failures lasting for only a short duration.

When we have a permanent site failure, other sites will take over as primary sites for data fragments being primaries at the failed site. This leads to lost transactions. The recovering site can identify all lost transaction with some help from sites that took over as primary site for some data fragments, while the sites taking over can identify transactions that could not be installed as candidates to be lost transactions. If a transaction could not be installed even though a 1-Safe commit was received, the transaction is a lost transaction. If a 1-Safe commit was not received no conclusion can be made about the transaction.

The two of the three different approaches I have considered to re-synchronize recovering data fragments with their corresponding current primary copy are similar in some parts. The difference between these two approaches, is where the compensation for lost updates is done. The first strategy lets the recovering site do all the compensation as the recovering site will have all lost updates in main memory. The other strategy uses the compensation done at the takeover site produced in the takeover process. This compensation is not complete, so some compensation will also have to be done at the recovering site. Catching up log produced after a takeover is identical in the two strategies.

5.2.4 Recovery with Global Compensation

The strategy with Global Compensation uses the compensation log already produced in the takeover procedure as part of the recovery of a failed primary replica. Assuming that not all of the log produced by primary replica before it failed, the compensation log already produced will not be enough to recover the failed replica.

After a take over, the node taking over as primary for some fragment will have to produce compensating log records for all lost updates which are updates that could not be installed at the backup according to requirements for installing updates. These updates belong to lost transactions or transactions that did not commit at the primary before the failure occurred. The compensation is necessary to maintain consistency of the fragment before it can accept new transactions. At the failed primary there will be produced more log than what was received at the backup before the failure. This means that the compensation done as a part of the take over procedure is not enough to restore the failed replica to a consistent state.

Remember the fact that a transaction can only be installed at the backup if all of its updates are received at the backup. This means that all the log not received before the failure, will belong to lost or uncommitted transactions. All this log must be compensated for before the failed replica can return to a consistent state.

When the failed site starts the recovery process, it will signal the rest of the system that

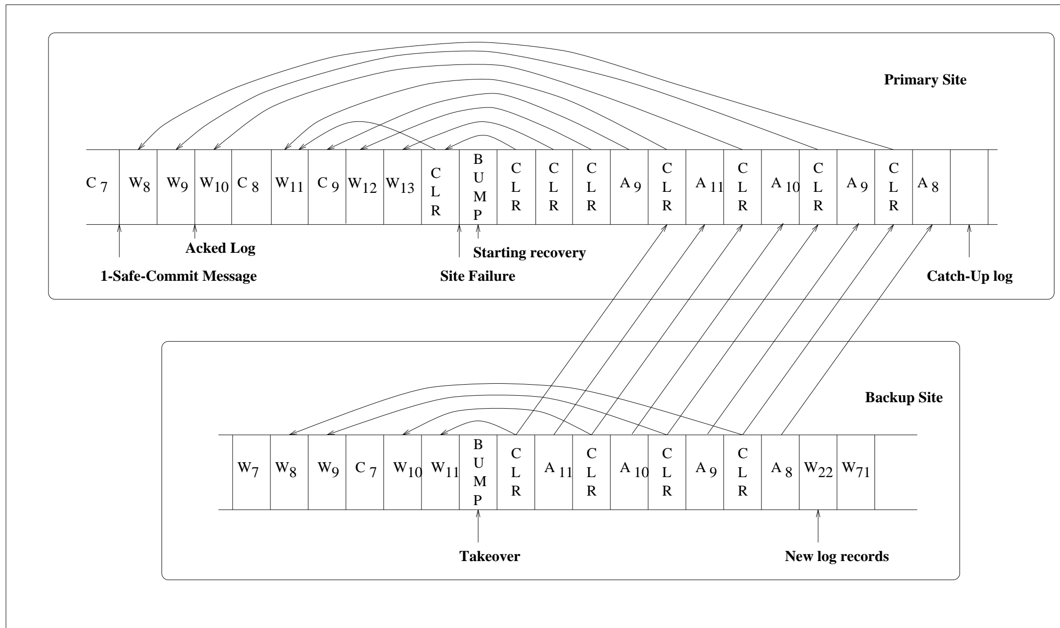

Figure 5.5: Recovery with global compensation.

it is recovering. The nodes that took over some failed replica will respond by sending compensation and catch-up log to the recovering site. The first log record sent will be a BUMP-UP record which tells where the log stream was cut in the takeover process. It contains the log sequence number (LSN) of the last log record received before the failure. The recovering site can now compensate all log that was not received at the backup as this part of the log stream will only contain lost updates. After compensating for all log records not received at the backup before the failure, the recovering site can now apply the compensation received from sites that took over. A recovery scenario is shown in Figure 5.5.

In the figure the different log records are:

- W for state changing operations such as INSERT, UPDATE, DELETE and delta-operations (increment and decrement).
- C for signaling COMMIT of a transaction locally at a site.
- GC for signaling that a transaction is successfully replicated at another site. This records is called GLOBAL COMMIT and corresponds to the COMMIT record when using 2-Safe replication.
- BUMP for signaling where the logical log stream was cut in a takeover.
- CLR for Compensation Log Record which is used to undo updates.
- A for signaling ABORT of a transaction both locally and globally.

This compensation will restore the fragment replica to a consistent state as it compensates for all updates violating the requirements for installing updates at the backup. Now the

recovering site can apply catch-up log which is log produced by transactions started after the take over. After the replica has applied all catch-up log it is said to be up to date and can become an ordinary hot stand-by replicate.

After becoming a hot stand-by replica, the replica can now signal to the current primary replica that it is ready to become primary once again. The current primary will upon receiving this message stop receiving operations from new transaction and announce itself as hot stand-by replicate. The recovered replica will after a short period of time announce that it has become primary again and the system returns to normal operation.

Detailed description of the **Global Compensation Recovery Algorithm** :

1. The algorithm starts with a forward redo scan of the last two checkpoints of log to be sure that all updates described in the log is reflected in the tuples of the fragment.

2. The recovering replica will signal that it is recovering. The current primary replica responds with sending the BUMP-UP record, compensation log produced in the takeover procedure and catch-up log produced after the takeover.

3. When the recovering replica receives the BUMP-UP record it will start to produce compensating log records for all the records that were not received at the backup before the failure:

   - For each W record a CLR which undoes this operation is produced.
   - For each R record nothing is done, but the LSN is incremented.
   - For each C record an ABORT record for this transaction is produced.
   - For each A record nothing is done, but the LSN is incremented.
   - For each CLR record a CLR is produced to undo this CLR. This means that a lost update will be restored. This update will be compensated for later in this undo scan or else its corresponding CLR will be received from the current primary replica.

   The reason for undoing all state changing operations not received at the backup, is to restore the recovering replica to a state logically identical to a possible state at the current primary. This state is equivalent to the state the current replica had before starting to produce compensating log records.

4. After all log not received before the failure have been compensated for, the recovering replica will start to apply compensation log and catch-up log received from the current primary replica.

5. When there is no more catch-up log to redo, the replica will become a hot stand-by replica. The replica can now become primary again so that the system can return to the data distribution described in the data dictionary.

How can the Global Compensation Algorithm guaranty a consistent recovery? To see why this algorithm will yield a correct recovery, we must assume that the takeover algorithm will bring the new primary replica to a consistent state before accepting new transactions. This was shown to be true in Section 5.1.5.

If the recovering primary replica compensates for all log not received at the backup before the failure, the log will be in the same logical state as the log stream at the new primary

replica before the takeover procedure started to produce compensation log for updates that could not be installed. By applying the compensation log produced at the new primary replica after takeover, the recovering replica will reach a consistent state logical identical to the current primary replica before excepting any new transactions. The applying catch-up log reflecting updates done to the new primary replica after takeover, the recovering replica will reach a state logically identical to the current primary replica.

5.2.5 Recovery with Local Compensation

A strategy where the recovering site do all the compensation for lost transactions will need almost no assistance from the operational sites. After the compensation is finished, catch-up of log produced by transactions started after the takeover will be similar to the algorithm with global compensation described above.

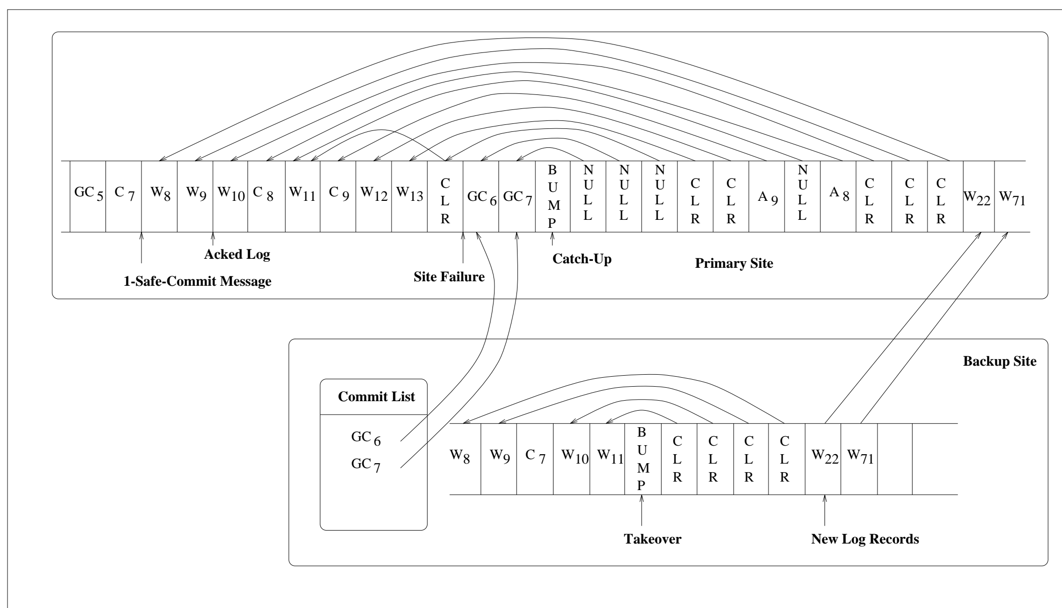

Figure 5.6: Recovery with local compensation

To avoid undoing transactions that were replicated successfully, the recovering site will have to ask the takeover site which of the transactions in the last 1-Safe message was replicated successfully. These transactions will put a global-commit (GC) record in the log like there is for all successfully replicated transactions.

The recovering site can now start the local compensation. Operations belonging to transaction with a GC-record in the log should not be undone, while all other operations belong to lost transactions or uncommitted transactions. A transactions without a GC-record in the log, but with a Commit record is a lost transaction and its operations are written to a special buffer to be merged, if possible, when the site have returned to normal operation.

There will have to be somewhere where the undo-scan may terminate because it will not find any operation belonging to lost or uncommitted transactions. If the transactions are

not allowed to live more than a certain amount of checkpoints, this could be used as a point for terminating the scan.

An obvious drawback of this strategy is the need of writing GC-records for all transactions when a 1-Safe message is acknowledged. The backup site will also have to maintain a list of transactions that are successfully installed, but this can be done as a part of the acknowledgment-message. There are also advantages about this strategy. The log garbage collector can use the GC-record as an indication to remove the whole transaction from the log, because it will not be needed for recovery. This will solve the problem of where to stop the undo-scan as the garbage collector can just scan the whole log.

The strategy with local compensation is shown in Figure 5.6.

Detailed description of the **Local Compensation Recovery Algorithm** :

1. The algorithm starts with a forward redo scan of the last two checkpoints of log to make sure that all updates described in the log is reflected in the tuples of the fragments.

2. The recovering replica will signal that it is recovering. The current primary replica responds with sending the global commit list which is acknowledge for installed transactions not received by the primary replica before it failed. The current primary will also send the BUMP-UP record and catch-up log produced after the takeover.

3. When the recovering replica receives the BUMP-UP record it will start to produce compensating log records for all the update records of transactions lost because they could be installed at the current primary after the failure:

   - For each GC record nothing is done but the corresponding transaction is registered as globally committed and should not have any of its updates undone. The LSN is also incremented.
   - For each W record a CLR which undoes this operation if a GLOBAL COMMIT record for this transaction have not been seen.
   - For each R record nothing is done, but the LSN is incremented
   - For each C record an ABORT record for this transaction is produced if a corresponding GLOBAL COMMIT record have not been seen.
   - For each A record nothing is done, but the LSN is incremented
   - For each CLR record nothing is done, but the LSN of the compensated record is registered and the LSN is incremented. (The compensated record will be found later in the undo scan and can also be ignored. The alternative would be to compensate for both the compensation and its corresponding non-CLR.)

   The reason for undoing all state changing operation not received at the backup is to restore the recovering replica to a state logically identical to a possible state at the current primary. This state is equivalent to the state the current primary replica had before start accepting operations from new transactions.

4. After all lost updates have been compensated for, the recovering replica will start to apply catch-up log received from the current primary replica.

5. When there is no more catch-up log to redo, the replica will become a hot stand-by replica. The replica can now become primary again so that the system can return to the data distribution described in the data dictionary.

How can the Local Compensation Algorithm guaranty a consistent recovery? We know that all log for some primary replica is kept until a successful replication have been acknowledged. This means that all updates done by lost transaction will be found in the log when a failed primary replica recovers. From the current primary replica the last successfully installed transactions is sent to the recovering replica. When the recovering replica know all successfully replicated transactions, one undo-scan of all log with undo information will wipe out all changes done by lost transactions. When this undo-scan is finished, the log will be in the same logical state as the log stream at the new primary replica had before excepting any new operation requests. By applying catch-up log reflecting updates done to the new primary replica after takeover, the recovering replica will reach a state logically identical to the current primary replica.

5.2.6 Scan Recovery

The two previous algorithms to recover failed primary replicas have used the log streams to re-synchronize the failed primary replica with the current primary replica of some fragment. These algorithms assume that the number of log records produced between takeover and recovery is much less than the number of tuples in the failed replica. For short down-times this is correct, but if a site is down for a long time like days, the nodes holding primary replicas may not have capacity to store all redo-log produced. The volume of produced log may also exceed a point where it is cheaper to produce a new replica at the failed site when it recovers. Remember that in order to recover a failed replica, the current primary replica must keep all produced redo log. Under normal operation this log can be deleted as soon as updates are successfully installed at another site.

The number of the log records produced does not necessarily mean that the same number of tuples have been updated. If the data in a fragment is fairly static, this can mean that a big fraction of the tuples in the recovering replica has not changed since the takeover even though the failed site may have been down for some time. From this fact it is clear that it would be cheaper to send only the changed tuples to the recovering replica instead of discarding the replica and produce a new one.

Scan Recovery is an algorithm where all tuples of the current primary replica are scanned to see if they have changed since the update. If they have changed, they are sent to the recovering replica where they are installed.

The tuples belonging to a failed primary replica may be in one of four different states when the failed site is recovering and trying to catch up with the current primary copies:

1. **last update received and not updated :** Transaction was not lost and the record have not been updated. The recovering site will have the latest copy.

2. **last update received and updated :** Transaction was not lost in the site failure, but the record have been updated and the recovering site will have to redo this log record.

3. **last update not received and updated :** Transaction was lost and the record have been updated. The recovering site will have to undo the lost transaction and redo the new update. This leads to a merging conflict.

4. **last update not received and not updated :** Transaction was lost, but the record have not been updated. The recovering will have the latest copy, but must undo the lost transaction.

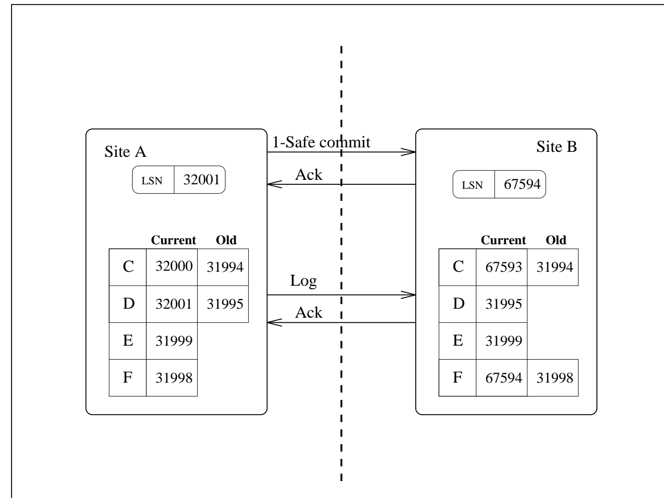

Figure 5.7: Different states of a record.

These different tuple states can also be seen in Figure 5.7. Most tuples will be in state 1 or 2. The fraction of tuples in state 2 will decide the cost of recovering the failed replica.

Re-synchronizing the recovering replica with the new primary replica can now be done by comparing the LSNs of the different tuples with the last LSN of the bump-up record. This scanning must be done after all compensation for lost records is finished. This procedure may look a bit like taking a snapshot of the fragment even though only a few log records may have to be sent to the recovering site. It can be used successfully when the site have been down so long that all log produced by transaction started after the failure had to be deleted because it consumed to much space on the disks.

Detailed description of the **Scan Recovery Algorithm** :

1. Like all other algorithm this one also starts with a redo scan of the recovering replica to be sure that all updates described in log records is reflected in the tuples.

2. Then the recovering replica will signal that it is recovering. The current primary replica responds with sending the BUMP-UP record and start scanning the tuples in the fragment. It will also start to keep the redo log to be applied after the scanning.

3. When the recovering replica receives the BUMP-UP record, it will know the LSN of the log record last received before the failure. To make the Scan Recovery work, the replica must restore itself to a state logically identical to a state the current primary have been in (not necessarily a consistent state). Therefore the recovering site will produce compensating log records for all log not received by the current primary replica before the failure. Updates by lost transactions may be compensated for at the current primary replica as part of the takeover procedure. These compensations

is not necessary to redo as the corresponding tuples will have LSNs greater than the BUMP-UP record and will be caught up in the scanning.

4. The changed tuples will be inserted as soon as they are received from the current primary, but no log will be produced for these inserts. When the last changed tuple is received, the recovering replica must also redo changes done after the scan began. After this, the replica become a hot stand-by replica and it is ready to take over as the primary replica again.

What will happen if the nodes crashes in the recovery process? The solution will be to start a new scan of the current primary replica. Note that there will not be need for any compensation during a second scan as the recovering replica already are in the state described in point 3. A problem concerning this algorithm is what to do with tuples deleted after the takeover, but before the recovery. This can be solved by letting deleted tuples remain in the fragment but marked as deleted. These so called tombstones will cause the corresponding tuple to be deleted when received by the recovering replica.

Chapter 6

Implementing 1-Safe in ClustRa

The ClustRa prototype is currently running with a 2-Safe replication strategy. The goals described in Chapter 3 have been reached regarding average response time and transaction throughput. The system is also capable of coping with node failures, but does not have the ability to withstand disasters like earthquakes which is necessary to reach class five availability. If the system should be able to continue after an earthquake, at least two replicas must be stored at two sites at geographically different locations. With the ClustRa prototype it is of course possible to place sites at geographically separated locations, but this would mean that the response time requirement is not met as about 10% of the transaction is expected to need replication in a typical telecom application. The transmission delay for a transaction needing replication to a far site would almost single-headedly exclude this transaction from finishing within the required 15 milliseconds.

It was decided to implement two solutions using 1-Safe replication to investigate if this could help the system to reach class five availability and at same time reach the response time requirement for 95% of the transactions. One of the implemented solutions is laid on top of ClustRa and the other is built into ClustRa.

6.1 1-Safe Solution Properties

The ClustRa project team will employ certain design goals or requirements when designing and implementing a 1-Safe disaster recovery mechanism to the ClustRa prototype. These goals are :

- **Scalability/Parallelism:** The system must be able to scale up to process a large number of transactions per second. At the hot stand-by site there should be parallelism in the installation process.
- **No Delay:** The disaster recovery mechanism should not introduce new delays to the normal processing.
- **No Take-over Delay:** End users should not experience any delay caused by system take-over.

A solution should use as much of the existing system as possible and introduce any new delays for transactions. Just as for the 2-Safe solution a 1-Safe solution should be able to scale almost linearly, have almost no takeover delay and not use more resources than a 2-Safe solution. A solution should also allow transactions to choose between 1-Safe and 2-Safe replication. The level of data availability should not be degraded compared to the current system using 2-Safe replication.

6.2 Replication with Autonomous Databases

The solution where the replication is placed on top of ClustRa was made rather simple. Replication was performed on procedure level by redoing the actual procedures at another site. The replication provided by ClustRa was turned off by having only one replica per fragment; the primary fragment.

It should be noted that this solution does not fulfill the implementation requirements described in the previous section. It can only be used when several assumptions can be made about the users and their access patterns. These assumptions will for instance allow the system to loosen up very strict consistency requirements and make the whole process of replication much simpler. Some applications can provide such assumptions and could therefore use this build-on solution with success. Note that even though this solution is simple, it does not necessary use less resources than a solution built into ClustRa.

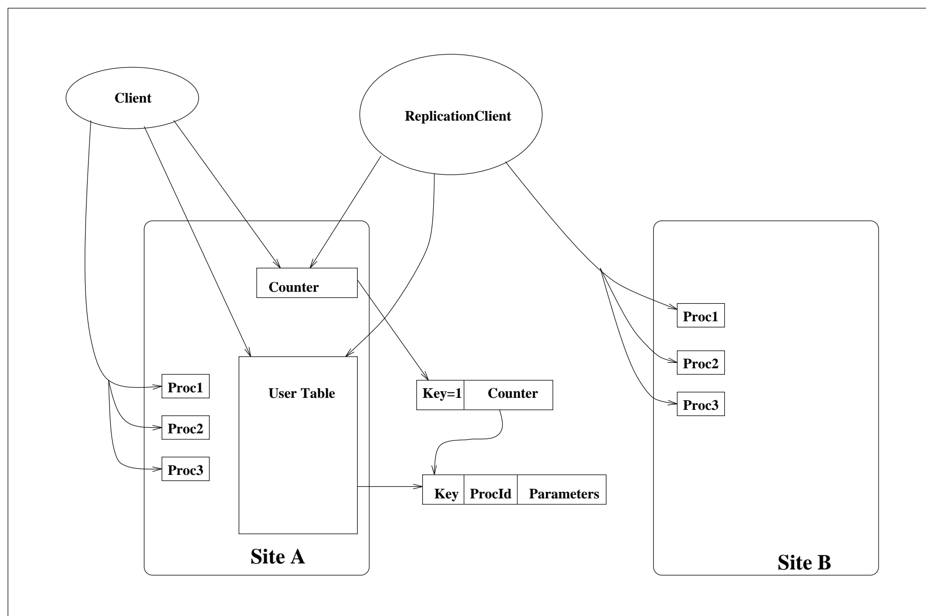

Figure 6.1: Replication laid on top of ClustRa.

The idea of the solution is shown in Figure 6.1. Ordinary user transactions will in addition to doing their operations also log their actions in a special User Table which is maintained only for the purpose of replication. There is also a Counter Table with only one tuple holding a counter. The assumption for this solution to work is that a client calls precompiled procedures named **Proc1**, **Proc2** and **Proc3** shown in the figure to execute operation

against the database. These procedures will after executing the operation requested by the client increment the tuple in the Counter Table. The current value of this tuple will then be used as primary key when they insert a tuple in the User Table telling which procedure was called and necessary parameters. This information will be sufficient to do a identical execution at some other site.

A special Replication Client will take care of redoing transactions at other sites. This client will poll the tuple in the Counter Table at some interval to see if there has been done any changes to the database. If this is the case, it will read the information stored in the User Table and call the same procedure with the same parameters at the other site. After receiving acknowledgment for this transaction, it can delete the corresponding tuple in the User Table.

In Section 6.4.4 the performance of this build-on solution is compared with the build-in solution and todays prototype running 2-Safe. The source code for the replication client can be found in Appendix B

6.3 Replication Inside ClustRa

Replication is a very important part of the ClustRa DBMS, so it makes more sense to let the system itself take care of replication instead of having dedicated replication clients as described in the previous section. The ability to work after a failure and to recover without any manual intervention is also an important characteristic of ClustRa.

The 1-Safe solution placed on top of ClustRa may not be able to offer functionality as takeover and takeback, because replication is done at procedural level. As this functionality is very important aspects of ClustRa, a 1-Safe solution built into the system seems more interesting.

The current 2-Safe strategy uses replication at tuple level. The log maintained for recovery is also used for replication. After studying the system, it seemed clear that very few changes would have to be done to implement a 1-Safe replication strategy.

The basic ideas of the proposed solution is shown in Figure 6.2. This figure can be compared with a corresponding transaction running with 2-Safe replication as shown in Figure 3.2. The idea is to let kernels and update channel work just as they would in a 2-Safe solution. The changes necessary to let the system run with 1-Safe replication is only applied to the Transaction Controller.

Remember that, with 2-Safe replication, the transaction controller (T0) asked all primary and hot stand-by slaves to participate in a 2-phase commit process (2PC). This resulted in inter-site communication and a waiting if the sites were far apart. With 1-Safe replication, only primary slaves, and hot stand-by slaves residing at the same site as the transaction controller, are asked to participate in the 2PC process. This means that the client can get a much earlier reply to its request. Even though the client is given a reply, the far site replication is not finished. Before the transaction terminates, it will send a 1-Safe message to its peer (T2) at the other site. This message will tell T2 which slaves on this site that have received updates from the transaction just terminated. With this information T2 will start a 2PC process involving the slaves having received log records from the particular transaction.

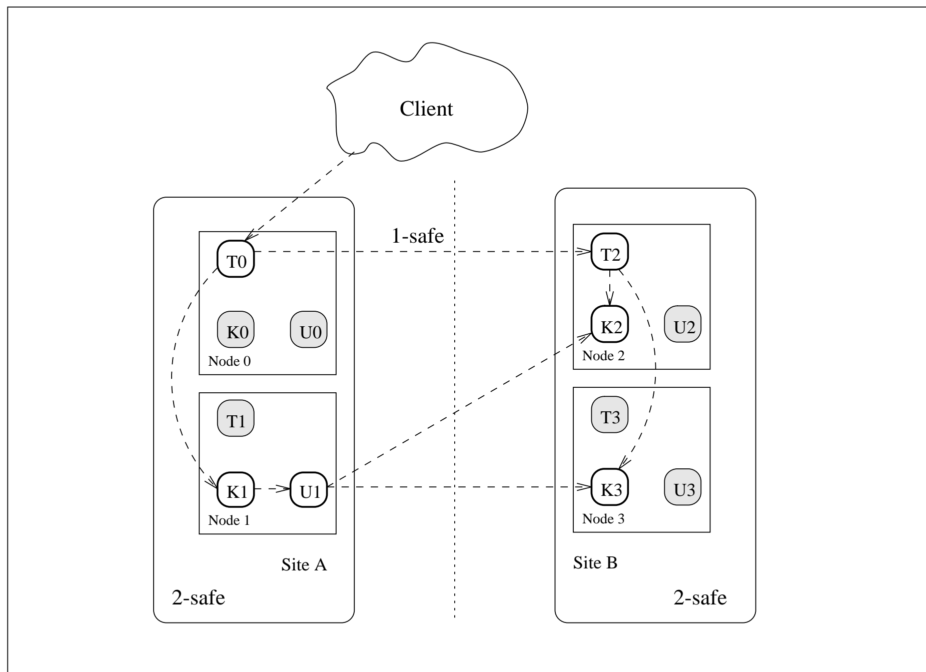

Figure 6.2: Executing a simple transaction using 1-Safe replication.

Note that in ClustRa, log to other sites for replication purposes is sent asynchronously over multiple parallel log streams. This is the reason why it is necessary to initiate a 2PC process at the backup site holding only hot stand-by replicas for the transaction in Figure 6.2. The 2PC process will give the necessary coordination for guaranteeing atomic installation of the transaction as described in the ACID rules.

6.3.1 The OneSafeChannel

An OneSafeChannel-object is introduced. This object contains functionality which allows the sites to be decoupled from one another with respect to time-dependencies. This means that commit processing at different sites is independent from one another. Transactions will commit at the originating site without waiting for replies or actions at another site. Only after the transaction is committed and the client is informed of the outcome, are the updates of the transaction installed at other sites maintaining replicas of the updated fragments.

The execution of a transaction will now differ a bit using the 1-Safe protocol between sites. Slaves and TCON-processes at other sites than the originating site, will not take part in the commit-processing. To distinguish between slaves at different sites, new fields are added to the transaction-object. The transaction object will have a *oneSafe* field which is true if the transaction is a 1-Safe transaction and false if it is a normal user transaction. Each slave in the transaction object will also have a new *onesafe* field (note that this field is different from the *oneSafe* field in the transaction object. A transaction object contains many slave objects). This field will be true if the slave is a 1-Safe slave (slave at another site);in which case it should not take part in the commit processing at the originating site. If the *onesafe* flag is false, it means that the slave is located somewhere at the originating site (either

primary or hot-standby slave) and should take part in the commit-processing.

The OneSafeChannel can be in six different states depending on outstanding requests, outstanding replies etc. This is illustrated in Figure 6.3.

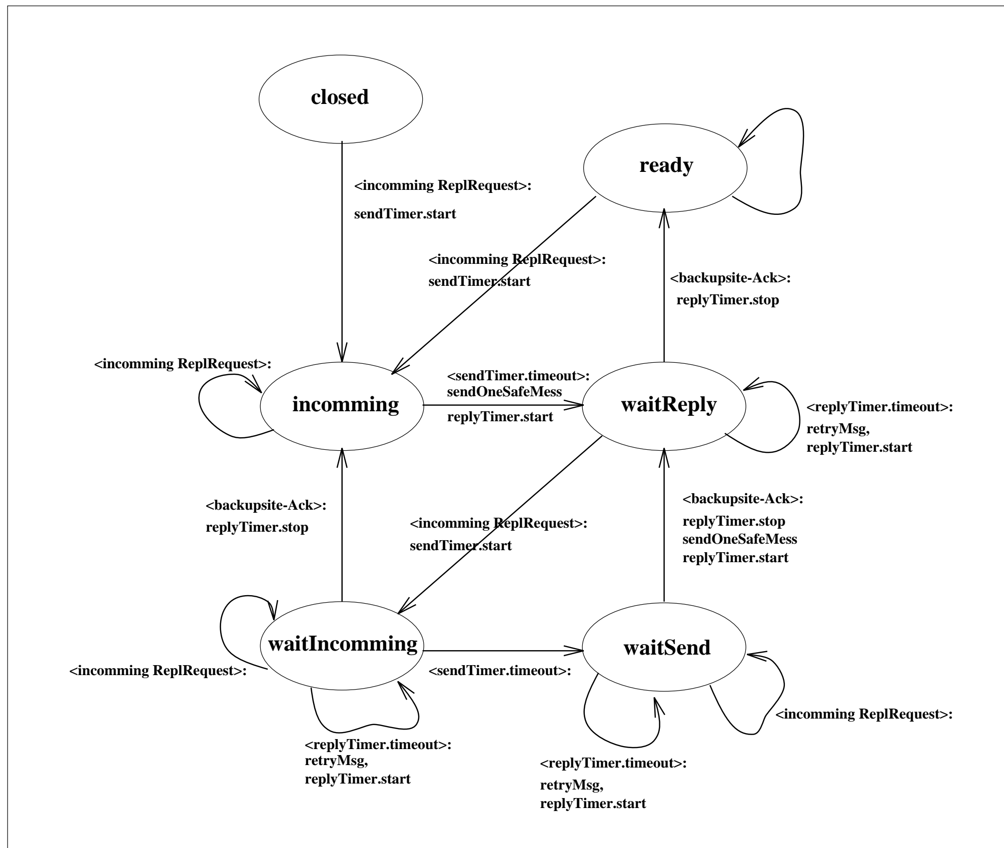

Figure 6.3: OneSafeChannel state machine.

When a transaction asks to be replicated to another site, it can change the state of the OneSafeChannel, start timers and the transaction will write its TransID into a list of outstanding requests before returning and committing as normal. The OneSafeChannel will build and send information about committed transactions to the corresponding TCON at the other site. This information is acquired by looking up transaction-objects in the list of outstanding transIDs. The OneSafeChannel state machine is shown at the next page.

Source code for the OneSafeChannel can be found in Appendix B

6.3.2 The OneSafe Message

After transactions terminate at a site, a message is sent to the other site with information about their hot stand-by slaves at this site. This information is sent in a OneSafeMessage sent by the OneSafeChannel. The OneSafeChannel will use a buffering strategy, so the message sent will contain information about several transactions. In the figure the layout of a OneSafeMessage and what information is put into TransactionMessage.

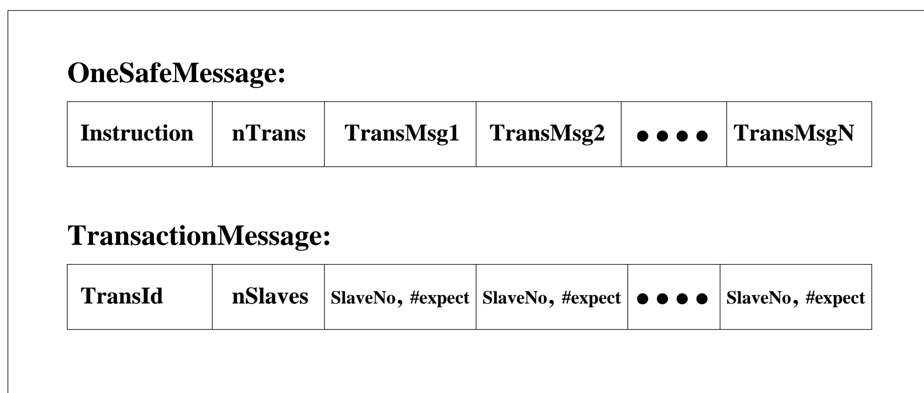

Figure 6.4: Layout of OneSafeMessage and TransMessage.

6.3.3 Transaction Execution

As mentioned earlier, slaves and TCONs at other sites than the originating site is not involved in commit processing of the transaction. Instead of involving these slaves and TCONs, the TCON at the originating site will tell the OneSafeChannel to take care of replicating updates done by some transaction to other sites. The actual sending of log records to other sites is carried out by the UpdateChannel as before. The OneSafeChannel on the other hand will send a "1-Safe commit" message to other sites with information about the number of updated records per slave. This message contains the information necessary to start a transaction at the other site with the same TransactionID as the original transaction. This transaction will be expecting the same number of records to be updated as the original transaction did.

Due to different distribution of data at the two sites, corresponding slaves at the sites might be expecting different number of records to be updated. The transaction at the backup site will commit in a similar manner as the original transaction. The only difference is that all slaves are treated as hot stand-by slaves (no primary slaves) and the transaction will have its **onesafe** flag set to true so that it will not ask to be replicated before it terminates.

The messages and events of a successful execution of a transaction is shown in the figure at the next page. This figure is made according to TCON state machine.

6.4 Measurements

A 1-Safe with replication inside ClustRa have been implemented according to the specifications described in Section 6.3. The choice whether the system should use 2-Safe or 1-Safe replication was left as a compile option. In this chapter measurements for both 1-Safe and 2-Safe with different level of replication will be presented.

The two different strategies will be compared with respect to response time and throughput. The response time is the most important criteria when comparing the two strategies, since the system demand real time response for its transactions. The transaction throughput is easier to adjust by adding more nodes to a configuration and thereby increasing the overall throughput. Remember that the throughput is expected to scale almost linearly with the number of nodes when the system uses multiple asynchronous log streams.

In the following sections the differences between 2-Safe and 1-Safe strategies will be studied. The performance tests gave numbers of throughput and response time, but it was really the difference between 1-Safe and 2-Safe under equal conditions that were most important to measure. Because the tests were run on not too powerful workstations, far from 95% of the transactions were able to finish within 15 milliseconds with moderate load. With heavy load the response time jump above 100 milliseconds, but then there will also be heavy resource contention. In a real system, resources will be dimensioned to provide real time response for the expected arrival rate of requests or there will be some sort of rejection of requests when the arrival rate gets to high.

All test results are given in tabular form in Appendix A.

6.4.1 Test Conditions

The tests were run on a configuration of four Sun SPARC 10 workstations, each playing the role of a database node. Two of the nodes were assigned as the primary site holding all primary replicas while the other nodes were assigned as backup site holding only hot stand-by replicas. Communication between the nodes were done through an ethernet.

The system parameters and test variables used during the performance measurements:

- The tables were initially loaded with 20.000 tuples. This was done to avoid any kind of blocking, deadlocks, and data contention.

- The number of replicas per fragment was varied from 2 to 4, with one replica as the primary and the rest as hot stand-by replicas.

- The delay of messages sent to logically different sites were set to either 5 milliseconds or none. The 5 millisecond delay corresponds to the distance between Oslo and Trondheim, a distance of approximately 500 kilometers. This distance is sufficient for having two sites with independent failure modes with respect to earthquakes and also other environmental disasters.

- For each measurement the system was run for a period of 150 seconds. The first 30 seconds was discarded to be sure that no transient behavior affected the measurement.

- In the test-runs, the number of session generating transactions varied from 1 to 20, divided on the two primary nodes. Each session generated transactions equivalent to an arrival rate of 10 transactions per second (TPS).

Two types of transactions were run on each set of test variables:

- 1-tuple transaction update one record in a table
- 4-tuple transaction update four records in a table. These transactions are TPC-B-like transactions. A TPC-B transaction does one insert and three updates, and is described in more detail in [Gra91].

The Test Programs

The programs generating the transactions, were able to vary the number of concurrent sessions running against the database. Each session generates 10 TPS. This is done by letting a session have a think-time of 100 milliseconds between each transaction generated. This think-time include the execution of a transaction. If the response time for a transaction is larger than the think-time, a new transaction is generated right away. This means that the throughput will increase with the number of sessions to a certain point where higher arrival rate only results in higher response time for each transaction.

To be able to measure the effect of having the sites at geographically different locations, the distance was simulated by delaying all inter-site messages. All these measurements must be taken with caution as they were effected by characteristics of the operating system. As long as the client generating transactions had a lot to do, the delay of messages was handled internally by the client. But at low arrival rates the client will ask the operating system to be suspended for some time if it does not have any work to do. The smallest time slot the operating system can allocate is 20 milliseconds. This resulted in messages being delayed 20 milliseconds instead of 5 milliseconds.

6.4.2 1-Safe versus 2-Safe with No Message Delay

Figure 6.5 shows the transaction throughput as the inter arrival rate is increased. As long as the arrival rate is low, the throughput grows linearly with the number of sessions. This means that there is enough resources to handle all requests. As the number of request increases the different curves come to a point where it is not possible to serve additional transaction requests.

The different curves represent the number of replicas maintained per fragment. In the figure it may look like the 1-Safe strategy have a higher throughput then the corresponding 2-Safe strategy. This is not necessarily true, as the 1-Safe strategy lets the backup-site do more of the work. In all the measurements it is the primary nodes that reach resource contention first.

The next two figures (Figure 6.6 and Figure 6.7) show the average transaction response time for 1-tuple and 4-tuple transactions as the inter arrival rate is increased. We see that both 1-Safe and 2-Safe have about the same transaction latency. This is as expected. When the arrival rate gets too high, the response time starts to increase linearly. This is an indication

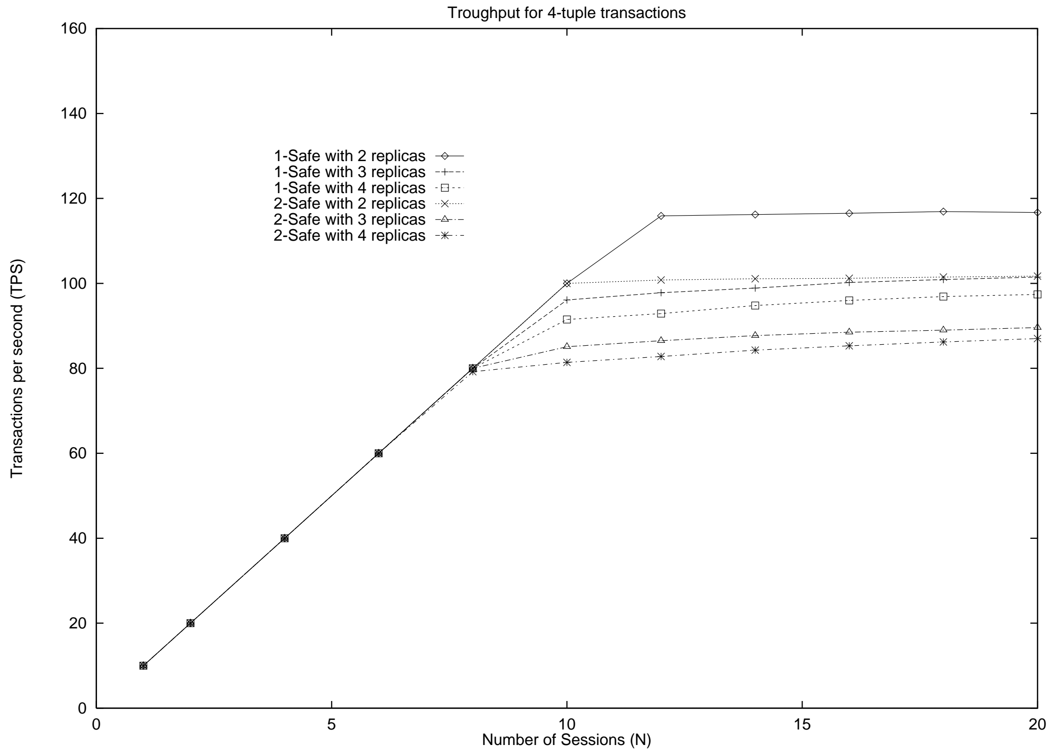

Figure 6.5: Throughput running 4-tuple transactions and no delay.

that additional request will only make internal queues longer and that transactions spends most of their time waiting for service by the CPU. Just before the response time start to increase linearly there is a sharp rise in transaction latency. This means that the system is not able to serve the transaction requests as fast as they arrive, resulting in queuing for service. In a real system there would be enough resources to get the average response time below the sharp rise in response time.

The lasts two figures (Figure 6.8 and Figure 6.9) in this section shows transaction latency plotted against the throughput. From about 80 TPS to 100 TPS the different curves rises very steep as the maximum throughput is reached. Only the lower part of the figure have acceptable response times. It is difficult to distinguish between the 1-Safe and 2-Safe strategies.

6.4.3 1-Safe versus 2-Safe with Far Site Replication

The throughput for transactions with delay of inter-site messages is the same as for transactions without this delay. As long as the response time stays below the think-time the throughput will be identical. When the response time gets higher than the think-time the system will almost have reached the point were the resource contention is the dominant factor with respect to throughput. Figure 6.10 and Figure 6.11 show transaction response time as the arrival rate is increased. The problem with the simulated message delay is quite visible for the 2-Safe strategy at low arrival rates. But it is clear that the 2-Safe strategy is affected by the distance between sites while the 1-Safe strategy is not affected.

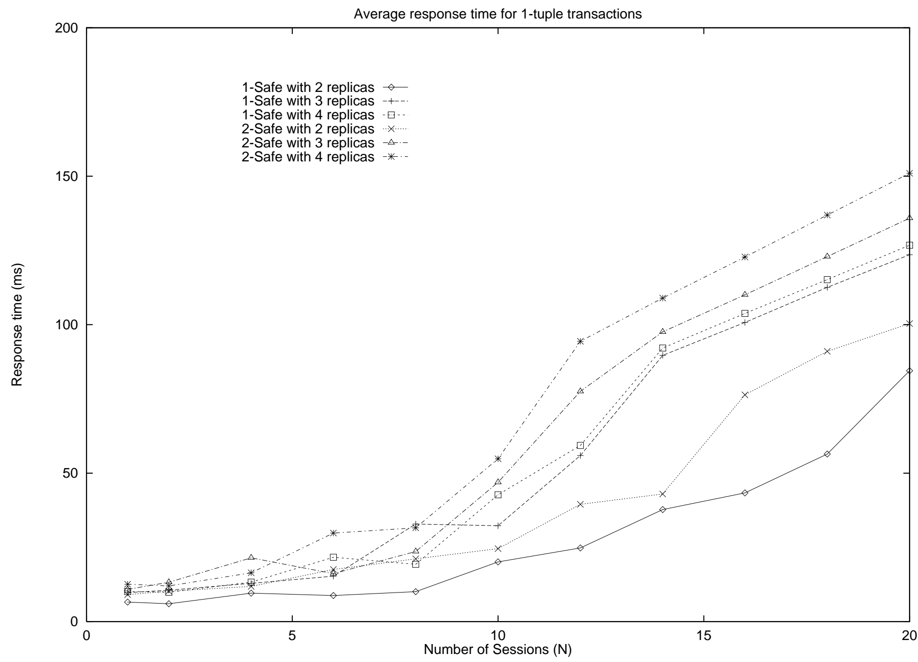

Figure 6.6: Response time running 1-tuple transactions and no delay.

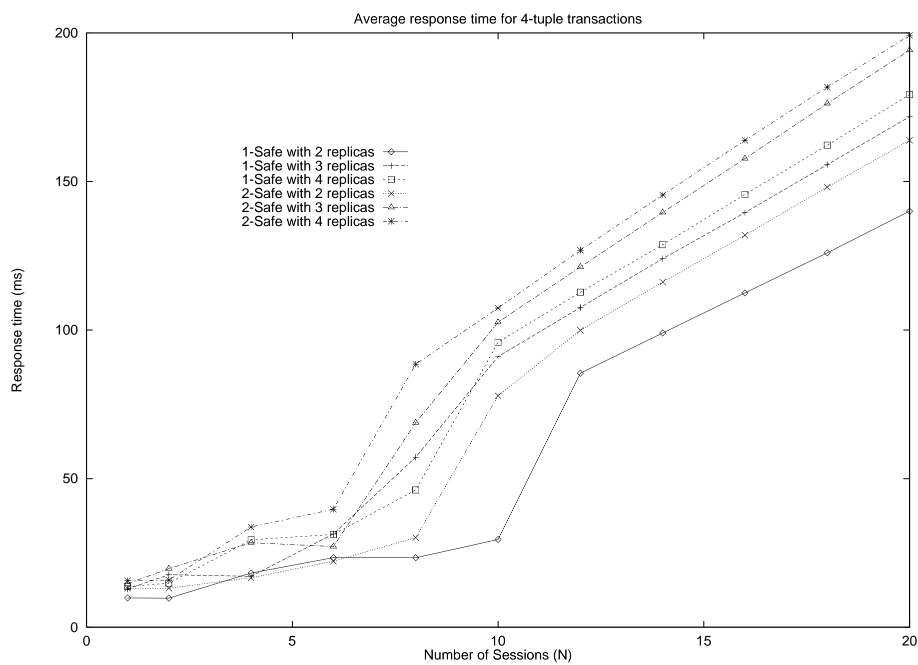

Figure 6.7: Response time running 4-tuple transactions and no delay.

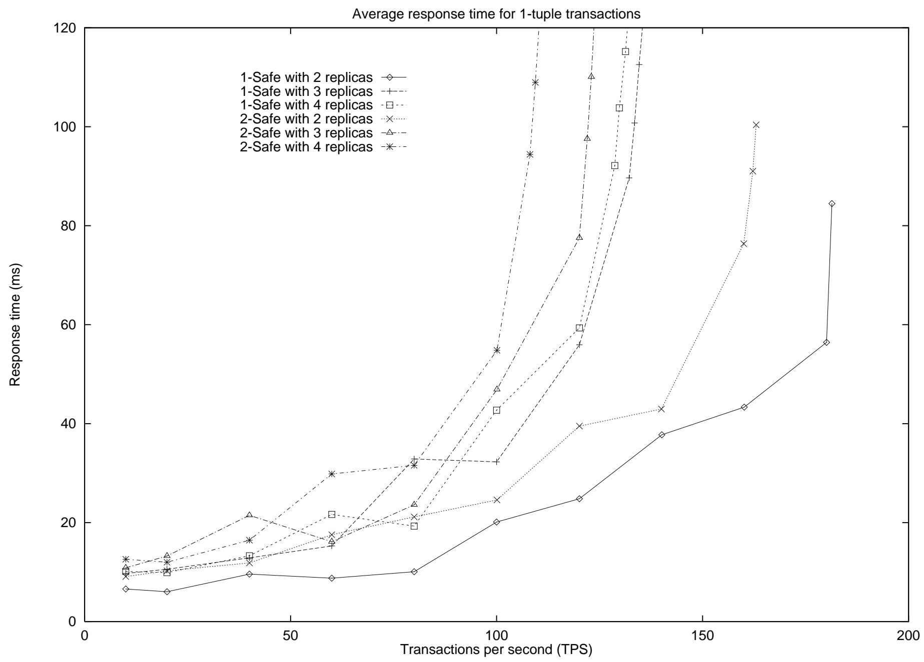

Figure 6.8: Response time running 1-tuple transactions and no delay.

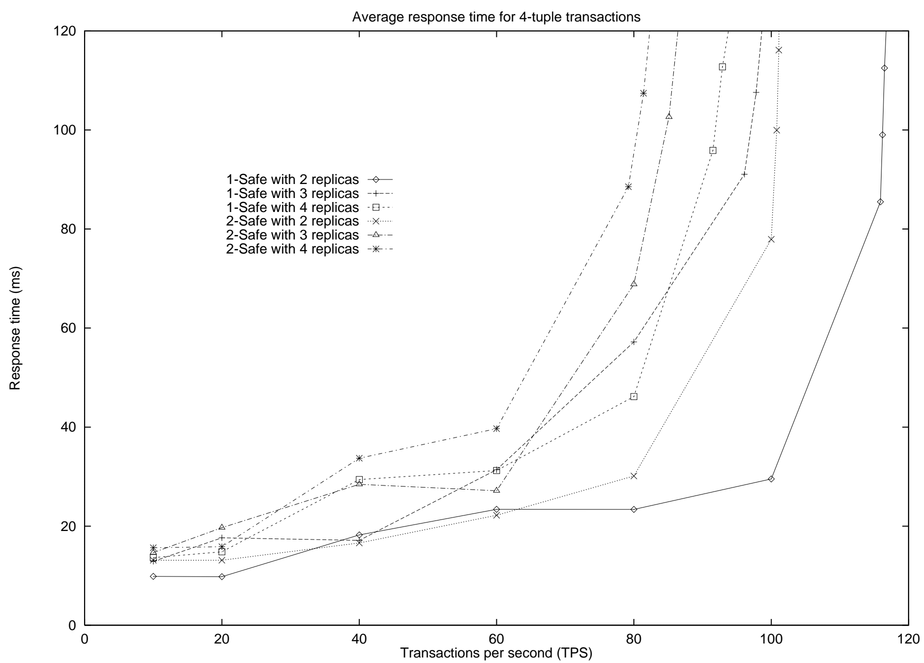

Figure 6.9: Response time running 4-tuple transactions and no delay.

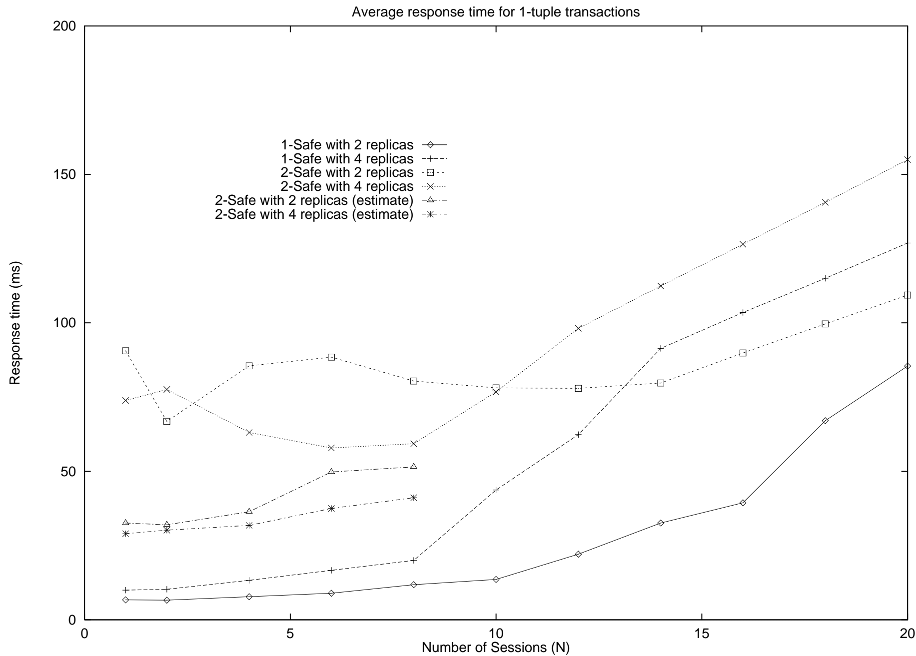

Figure 6.10: Response time running 1-tuple transactions and 5 ms delay.

In the figures the predicted minimum and average latency is also plotted for 2-Safe transactions. These predictions is made by taking the corresponding values for 2-Safe transactions without any delay and adding 20 milliseconds which is the total message delay for a transaction (two round-trips). We see that even with these predictions, the transactions will not be able to finish within their time limit.

Figure 6.12 and Figure 6.13 also show response time but plotted against the transaction throughput. From the figures we see that the effect of delaying inter-site messages diminishes as the throughput gets closer to the point of resource contention. After the system have reached this point, delaying messages will have no effect. The reason for this is that the resource contention will become the dominant factor with respect to response time and the effect of not delaying messages would only result in longer waiting-time in some queue waiting for service from the CPU.

6.4.4 Build-On Solution

The solution where replication is done outside ClustRa was made rather simple. Therefore this solution can only run 1-tuple transactions, without big changes being done to the programs. In addition, it is only possible with node to node replication. Only a few performance runs where done to compare this method with the 1-Safe and 2-Safe solutions described above.

The performance tests showed that the build-on solution could run 140 TPS in burst mode and 80 TPS over some time. The corresponding values for the other strategies over time is 180 TPS for 1-Safe and 160 TPS for 2-Safe. With further development it will be possible to increase the performance. Today each user transaction results in four requests to the

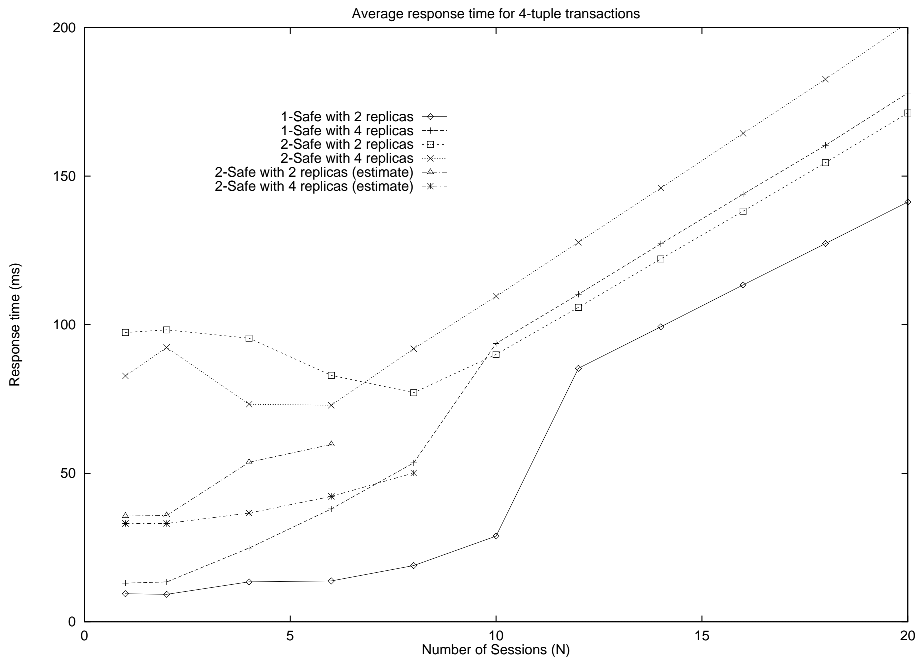

Figure 6.11: Response time running 4-tuple transactions and 5 ms delay.

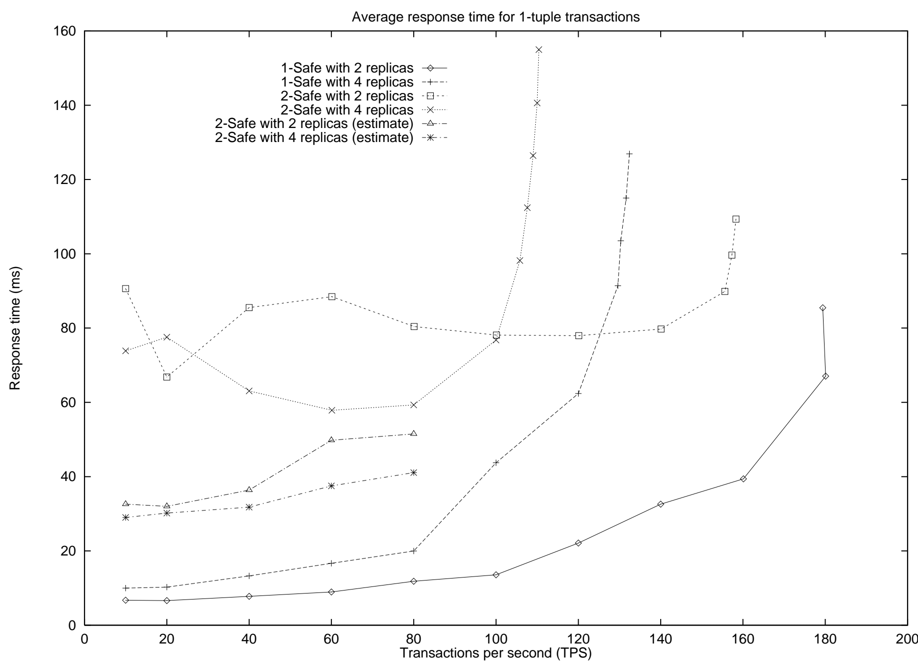

Figure 6.12: Response time running 1-tuple transactions and 5 ms delay.

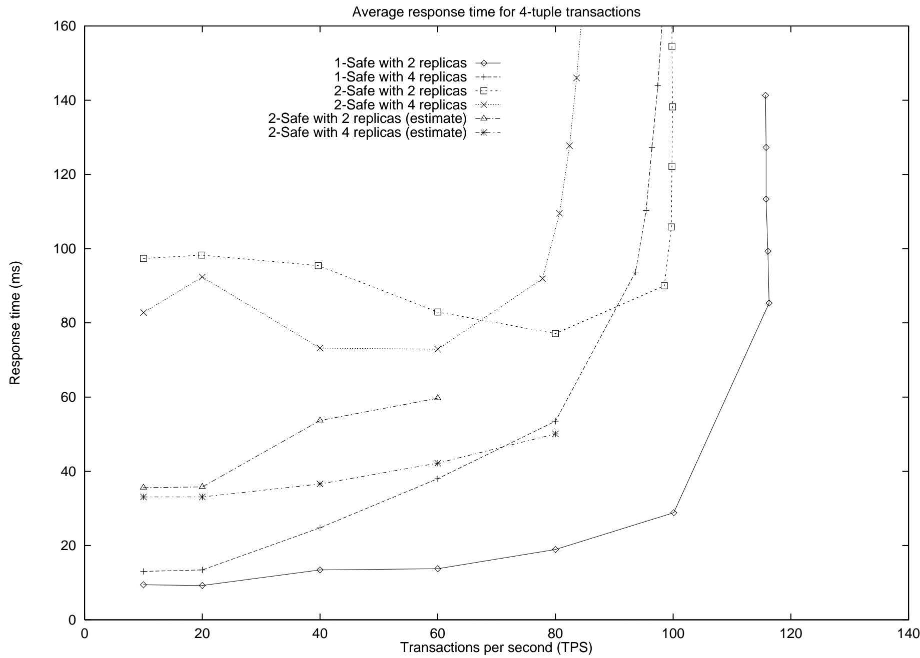

Figure 6.13: Response time running 4-tuple transactions and 5 ms delay.

database. Some of these requests can be grouped and thus save both communication and CPU costs. This will increase throughput. It should be noted that the build-on solution does not provide the availability and reliability offered if the replication is done within ClustRa.

The average transaction latency with delayed inter-site messages was a bit lower than for the 1-Safe strategy. The reason for this is that the current build-on solution only offer node to node replication. The transactions will have the advantage of only accessing one node locally per transaction.

6.4.5 Conclusion

It is clear that the 1-Safe strategy outperforms the 2-Safe strategy when it comes to far site replication. This was as expected when the nature of the two strategies is compared. By using a 1-Safe strategy for replication between sites in ClustRa, the system will be able to provide real time response for update transaction as well as read-only transactions.

The difference between 1-Safe and 2-Safe was 133% at low arrival rate (10-20 TPS) and 66% at 60 TPS in arrival rate.

Even though the simulation of the transmission delay did not work too well, it became quite clear that the 2-Safe strategy was effected by the delay while the 1-Safe strategy was not.

Chapter 7

Conclusion

This thesis have shown that a 1-Safe execution scheme can be successfully adopted in ClustRa. Measurements show that a 1-Safe strategy can provide real time response for state changing transactions which is not possible with a 2-Safe strategy. It is also possible to mix transactions running 1-Safe and 2-Safe.

A 1-Safe strategy with local and far site replication, will also provide the same availability as a 2-Safe strategy with respect to single node failures and total site failures. The availability is lower for 1-Safe with respect to multiple node failures.

Due to problems with simulating transmission delay to logical distant sites, it has not been possible to accurately quantify the improvement in response time when using a 1-Safe execution scheme. To quantify the improvement in response time, minimum estimates was made for the 2-Safe test runs. These estimates shown that the 2-Safe execution have a response time that is between 66-133% higher than a corresponding 1-Safe execution running TPC-B-like transactions.

Even though the tests look promising for the 1-Safe execution scheme, there are problems that must be solved. The existing solution need read operations, in contrast to 2-Safe, to appear at the backup to guaranty consistency. In a typical telecom application the ratio of update transactions may be lower than 10%. If all update and read operations must be sent to the backup, this will introduce very high costs.

Future work will be to implement functionality for takeover after failures and re-integration after repair. There will also be a need for distinguishing read operations belonging to update transactions from operations of read-only transactions.

Appendix A

Measurements Data

Appendix B

Source Code

Bibliography

[Gra92] Grey, J. N., Reuter, A., 1992. "Transaction Processing: Concepts and Techniques.". Morgan Kaufmann Publishers, San Mateo, California, USA.

[Gra91] Grey, J. N., editor 1991. "The Benchmark Handbook for Database and Transaction Processing Systems". Morgan Kaufmann Publishers, 334 p.

[GarPol94] Garcia-Molina, H., and Polyzois, C. A., 1994. "Evaluation of remote backup algorithms for transaction-processing systems". ACM Transaction on Database Systems 19, 3, p.423-449.

[GarPol90] Garcia-Molina, H., and Polyzois, C. A., 1990. "Issues in disaster recovery". In IEEE Compcon. IEEE, New York, p.573-577.

[GPH90] Garcia-Molina, H., Polyzois, C. A., and Hagmann, R., 1990a. "Two epoch algorithms for disaster recovery". In 16th International Conference on Very Large Data Bases. VLDB Endowment, p.222-230.

[GPKH90] Garcia-Molina, H., Polyzois, C. A., Halim, N., and King, R. P., 1990b. "Overview of disaster recovery for transaction processing systems". In IEEE 10th ICDCS. IEEE, New York, p.286-293.

[GPKH91] Garcia-Molina, H., Polyzois, C. A., Halim, N., and King, R. P., 1991. "Management of a remote backup copy for disaster recovery". ACM Transaction on Database Systems 16, 2, p.338-368.

[MoTrOb93] Mohan, C., Treiber, K., Obermarck, R., 1993. "Algorithms for the management of remote backup data bases for disaster recovery". In IEEE 9th International Conference on Data Engineering. IEEE, New York, p.511-518.

[BurTre90] Burkes, D. L., Treiber, K., 1990. "Design Approaches for Real-Time Transaction Processing Remote Site Recovery". In IEEE Compcon. IEEE, New York, p.568-572.

[Gue91] Guerrero, J., 1991. "RDF: An Overview". Tandem Systems Review,7(2), October 1991.

[Syb92] SYBASE Corp., 1992. "Replication Server Overview". SYBASE System Documentation.

[Lyo90] Lyon, J., 1990. "Tandem's Remote Data Facility". In IEEE Compcon. IEEE, New York, p.562-567.

[Lyo88] Lyon, J., 1988. "Design considerations in replicated database system for disaster recovery". In IEEE Compcon. IEEE, New York, p.428-430.

[JLRS94] Jagadish, H. V., Lieuwen, D., Rastogi, R., and Silberschatz, 1994. “Dali: A high performance main memory storage manager”. In Proceedings of the 20th International Conference on Very Large Databases, p.48-59.

[HLNW94] Heytens, M., Listgarten, S., Neimat, M.-A., and Wilkinson, K., 1994. “Smallbase: A main-memory DBMS for high-performance applications”. Research report, Database Technology Department, Hewlett-Packard Laboratories.

[HTBHG95] Hvasshovd, S. O., Torbjørnsen, Ø., Bratsberg, S. E., and Holager, P., Galindo-Legaria, C. A., 1995. “An Overview of the ClustRa DBMS”. ClustRa DBMS System Documentation.

[HvToBrHo95] Hvasshovd, S. O., Torbjørnsen, Ø., Bratsberg, S. E., and Holager, P., 1995. “The ClustRa Telecom Database: High Availability, High Throughput, and Real-Time Response”. Submitted to VLDB ’95, Zurich, Switzerland.

[HvTo95] Hvasshovd, S. O., Torbjørnsen, Ø. “A Performance Study of ClustRa”. TF R 24/95. ClustRa System Documentation, May 1995.

[Agr95] Agrawal, D., El Abbadi, A., Jeffers, R., Lin, L. 1995. “Ordered Shared Locks”. VLDB Journal,4, p.87-126.

[Inf94] INFORMIX-OnLine Dynamic Server,Database Server,Administrator’s Guide, Volume 2, Version 7.1. Informix Software,Inc., 4100 Bohannon Drive, Menlo Park, CA, USA, December 1994.

[Tre95a] Trettenes, T. A., 1995. “One-Safe Update in Distributed Real Time Data Bases”. Project report in course 45073 Computer Science, projects. Published by the local printer, NTH, Norway.