The ADR Replication Manager

Rainer Gallersdörfer Matthias Jarke Matthias Nicola

RWTH Aachen, Informatik V, Ahornstr. 55, D-52056 Aachen, Germany
{gallersd,jarke,nicola}@informatik.rwth-aachen.de, Fax: +49-241-8888-148

Key words: replica management, distributed databases, transaction processing, asynchronous update propagation, relaxed coherency, performance evaluation, telecom applications.

Abstract

ADR (Atomic Delayed Replication) is a controllable replication manager implemented on top of commercial distributed relational databases. ADR's goal is to enable various well-defined trade-offs between database coherence, throughput and response time in large database networks, e.g. for telecom applications. By combining a strategy for distributed database design with a specific replication protocol, ADR preserves the ACID properties with a controlled relaxation of coherence between primary and secondary copies. We first discuss formal characteristics of ADR, and present the implementation techniques required to realize these formal characteristics on top of commercial distributed database technology. Then, after reviewing a validated analytical performance model for the approach, we demonstrate its flexibility by summarizing experiences with two industrial ADR applications in telecommunications management, both jointly developed with Philips Laboratories. One is database support for the integrated operation and evolution of Intelligent Network telephone services, where secondary copies are held within a distributed database system optimized for throughput and availability during schema evolution. The other concerns database support for mobile phones in a City-wide DECT setting (*Digital Enhanced Cordless Telecommunications*), *where secondary copies are held in main memory caches outside the DBMS.*

1 Introduction

It is widely recognized that database technology must become more flexible to deal with the challenges of novel and widely different application demands, without resorting to expensive and hard-to-maintain special purpose implementations. On the other hand, database solutions for such application domains should not only reuse existing commodity database systems with as little overhead as possible, but should also preserve as much as possible the theoretical properties developed in the database community to ensure safety and predictability.

In telecom databases, the domain that motivated the work reported here, the problem of developing and using adequate database technologies has been characterized by a fundamental trade-off between consistency, performance and distribution, as depicted in Figure 1 [Kerboul 93]. Consistency and performance are classical database requirements while distribution is the essence of many modern applications like intelligent network services or mobile applications. Design options can, at a clearly simplified level, be seen as points or small areas within the cube shown in figure 1. Central database management systems provide consistency and reasonable performance but

obviously no distribution. Classical distributed database systems allow for both distribution and full consistency, but the use of synchronous two-phase commit protocols (2PC) severely impairs performance. The Internet can be seen as a distributed information system that trades consistency for throughput and distribution. The "ideal", distributed, fully consistent, real-time commodity database system at low costs seems out of reach.

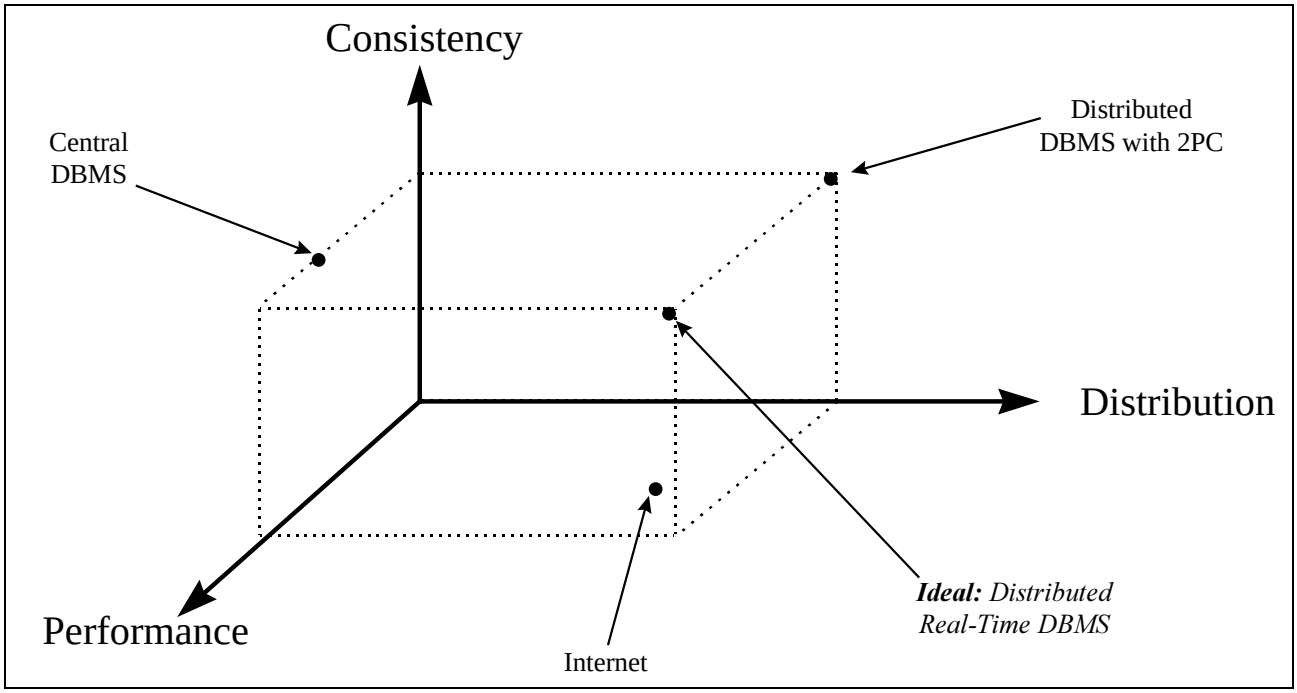

Figure 1: Trade-Offs in distributed database technology for telecom applications [Kerboul 93].

When designing alternative solution points in the space sketched in figure 1, strategies for data replication and transaction management obviously play a central role. Several database vendors are offering replication management products, and many more strategies for replica management have been proposed in research. However, the precise trade-off between consistency, distribution, and performance remains difficult to characterize in many of these approaches.

In the INDIA project, jointly conducted by Philips Laboratories and the Information Systems group at RWTH Aachen from 1993-1997, we initially studied precise characterizations of relaxed coherence between data replicas, and developed analytical models for the impact of coherence relaxation on performance [Gallersdörfer, Nicola 1995]. This was motivated by database needs of a design and management environment for Intelligent Network (IN) services Philips had developed.

Based on these results, we have designed and implemented the ADR replication manager, a practice-oriented extension of commercial distributed (relational) databases which combines a controlled strategy for replication management with an extended approach to distributed database design, adding a grouping operation between the traditional steps of logical database fragmentation and physical allocation. More precisely, ADR was designed to satisfy the following properties:

- ADR runs on top of existing relational database technology, without any need for modification on the commodity database systems, but its core implementation is still independent from any specific database system.
- ADR employs asynchronous update propagation to improve performance while retaining global conflict serializability – one of the most important distinguishing factors with respect to existing replication managers.

- ADR prevents unbounded aging of secondary copies and guarantees defined levels of data accuracy, based on the model presented in [Gallersdörfer, Nicola 95].
- ADR is flexible and widely applicable because not only „conventional“ replicas but also main memory replicas can be managed, thus allowing tuning for response time as well as throughput performance.
- ADR allows for dynamic evolution of the database and replication schema, i.e. the relational schema and the number and placement of secondary copies can be altered while the system remains in full operation.

Of course, there are also limitations of the approach which are acceptable in the telecom domain we have been focusing on but may not be acceptable in other domains. In particular, a main limitation is that ADR will work well only if the read operations in transactions are limited to reasonably small and well-defined parts of the database. “Read operations” also includes especially the integrity checking operations of the database which in our approach serve as the basis for the “grouping” operation we add to distributed database design. In telecom applications like Intelligent Networks and value added call management, most database activities concern data related to two parties of a phone call. These are small partitions of data and there are no integrity constraints spanning multiple such partitions. Hence, ADR is well suited for this application. Another group of applications that will work well with ADR are those in which transaction splitting as proposed by [Shasha 92] is acceptable. The limitations of ADR are not acceptable in applications which focus on global integrity conditions and the analysis of large data sets, such as OLAP applications.

Two commercial applications using ADR have been implemented in cooperation with Philips in 1995/96. Extensive measurements on these systems confirm the analytical results [Gallersdörfer et al. 97]. In the first application, ADR provided database support for the IN design and operations environment developed by Philips Laboratories (meanwhile patented and commercialized by another company). This implementation demonstrated mostly the controlled schema evolution, throughput and scalability improvement in a setting where all database replicas run under the Sybase distributed database management system. The second application provided DBMS support for mobile telephony in a city-wide DECT environment, developed jointly with a French Philips subsidiary (meanwhile sold to another global player in the telecom market). In this application, it was demonstrated that ADR can also support replicas in main memory (without special main-memory DBMS software), thus satisfying the extreme response time demands of the application. In addition, while ADR essentially follows a primary-copy approach and focuses on management of secondary copies, this application showed that it could easily be combined with an approach that had hot standby primary copies for ensured availability.

This paper is a comprehensive description of ADR and the experiences gained with it. In section 2, we present the basic idea of the approach and compare it to related work on replication management in research and industry. In section 3, we first formally define ADR and show that it preserves global serializability, with the possibility for read transactions accessing aged data to a degree bounded by the coherence index defined for the application. This demands three specific properties from the ADR system which pose some tricky implementation challenges. The rest of section 3 shows how active database technology and metadata management in the database itself have been used to solve these implementation issues. Finally, section 4 – after briefly recapitulating a slightly improved version the analytical performance model from [Gallersdörfer, Nicola 95] – reports on the two application experiences mentioned above as an evaluation of the approach. Section 5 summarizes the main results and points to a number of extensions currently under development.

2 Overview and Related Work

The general idea of ADR is to combine replication management with distributed database design. ADR provides means to define and alter the database and the replication schema, as well as mechanisms for atomic but asynchronous and possibly delayed propagation of updates. In sections 2.1 and 2.2 we describe how ADR is based on an application oriented partitioning of data so that different levels of consistency can be defined and maintained. Section 2.3 gives an overview of the general ADR system architecture, and section 2.4 compares our approach with related work in replication management and telecom databases.

2.1 Data Partitioning

Replication is based on the primary copy approach where replicas (primary and secondary copies) are defined on the granularity of so-called *partitions*. Partitioning requires an additional database design step named *grouping* as an intermediate step between fragmentation and allocation, as depicted in figure 2. Data fragments which are logically closely related either because of integrity constraints or because of frequent joint usage are grouped together to partitions which represent the units of allocation, and thus replication.

The aim is to define partitions such that most consistency requirements are partition *internal*. Transactions are defined to consist of *n* read-steps and at most one write step such that different steps access logically independent data items. A skilful partition schema should then allow a major share of the transaction steps to be executed on single partitions. Restricting transactions to a maximum of *one* write step is a crucial requirement for the consistency properties guaranteed by ADR.

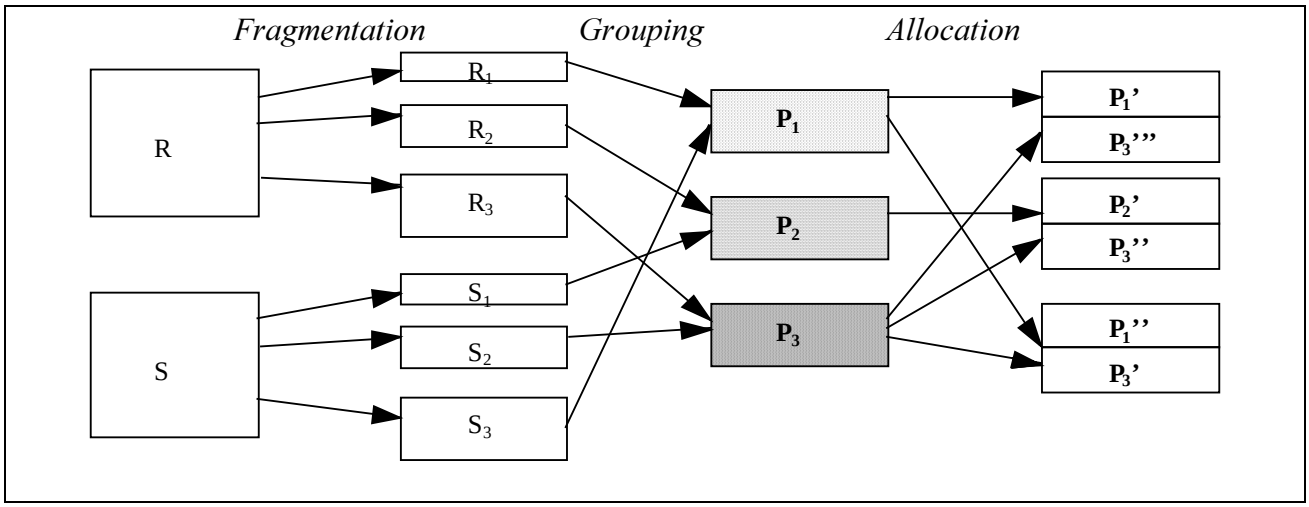

Figure 2: Partitioning as a database design step

The example below shows a (slightly simplified) transaction related to a phone call using a virtual private network (VPN) service. The transaction splits into a write-step and a read-step. In the write-step, a counter variable (CNT) for the service 600 of subscriber 1234 is increased for statistics and billing. The read-step is used to find the real world telephone number assigned to the short number SN_11 that has been used to initiate the call. Obviously, the operation of increasing the counter is independent from the value of the phone number read. Still, both steps have to be within the context of a single transaction to provide atomicity: an abort during the read-step also requires to undo the write-step because we do not want to charge users for calls which could not get connected.

```
BEGIN TRANSACTION
   BEGIN WRITE STEP „S.600.1234“
     UPDATE variable SET value = value + 1
       WHERE serv_nr = 600 AND subs_nr = 1234 AND var_name = CNT

   BEGIN READ STEP „U.600.1234“
     SELECT value FROM variable
       WHERE serv_nr = 600 AND subs_nr = 1234 AND var_name = SN_11
COMMIT TRANSACTION
```

Figure 3 shows the partitions used by the sample transaction. The operational part of the VPN service owned by subscriber 1234 is defined by a single record in the table *Subscriber* and by the VPN numbers (SN_11, SN_12 and SN_13) in the table *Variable*. These records are grouped to a partition „U.600.1234“. The statistical part of the same VPN consists of the counter record in the table *Variable* and a description record in the table *Service* which form another partition named „S.600.1234“. The partition „U.600.1234“ will rarely be changed but read very often. Therefore it is clever to replicate it. Partition „S.600.1234“ should not be replicated because it is often modified.

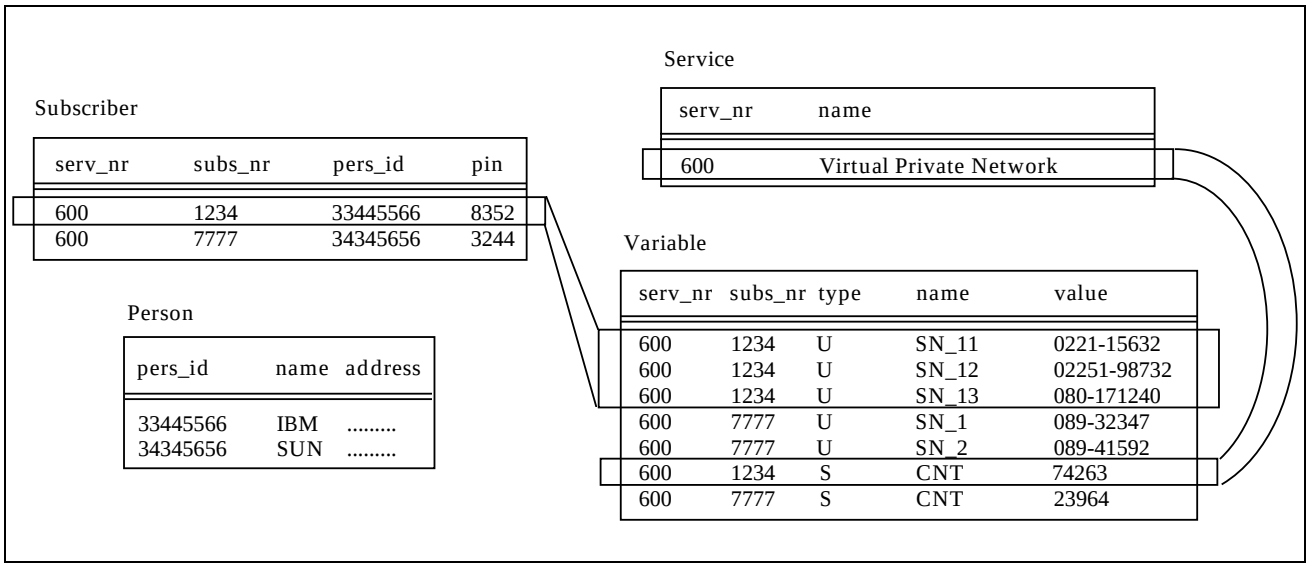

Figure 3: Partitions for a VPN service in the intelligent network

Structuring partitions and transactions this way is surely not possible for every imaginable database application. However in many OLTP applications with short transactions over few records, the ADR idea can be employed. As the example indicates, we initially implemented and tested ADR in a distributed database system by Philips supporting *Intelligent Networks* (IN).

2.2 Internal and external consistency

Partitions are replicated according to the primary copy approach, i.e. there is one primary copy and *m* secondary copies. This approach is followed by most replication algorithms; its theoretical merits have been recently argued by [Gray et al. 96]. Updates are propagated from primary to secondary copies asynchronously, i.e. not within the context of the original update transaction and possibly delayed. Hence, secondary copies may age but still provide a sufficient level of *partition internal consistency* that the application is satisfied with.

Definition: A partition is called *internally consistent* if all partition internal consistency requirements are fulfilled.

Definition: A set of partitions are called *externally consistent* if all (internal and global) consistency requirements are fulfilled.

The ADR system ensures that the set of primary copies is always externally consistent; secondary copies are always internally consistent but may be out of date. Furthermore, a transaction's write-step (if any) always has to be executed on the primary copy, perceiving (and preserving) external consistency. Read-steps can be carried out on any secondary copy, as long as the application is satisfied with internal consistency. Otherwise read operations have to be included in a write-step. For instance, imagine a transaction that consists of two read steps reading two different secondary copies SC1 and SC2. ADR guarantees that both read steps see a state of SC1 and SC2 respectively, which once was a valid state of the respective primary copy. Yet, the states of SC1 and SC2 may be of a different age such that SC1 and SC2 may reflect a combination of values which never existed among the related primary copies. If consistency regarding *both* partitions (*partition external consistency*) is required, then the two reads have to be embedded in a write step. This forces the read operations to be executed on the primary copy.

Since every transaction has at most one write-step, atomicity of an update transaction can be ensured by a single site, i.e. the site holding the primary copy. ADR does not require distributed concurrency control but the possibly distributed read-steps can be executed under local concurrency control at the speed of a centralized DBMS. The synchronous two phase commit protocol is only used in the rare case that a write step needs to access multiple partitions which are located at two or more different sites.

2.3 Overall system architecture

ADR has been implemented on top of commercial relational database technology, namely Sybase SQL Server. The general system design is depicted in Figure 4. Applications access databases through the ADR module. To avoid hindering the database's communication parallelism to the application site, and to minimize the communication overhead between the application and ADR, ADR is not running at the database site but at the application sites where it is a software library linked to the application source code.

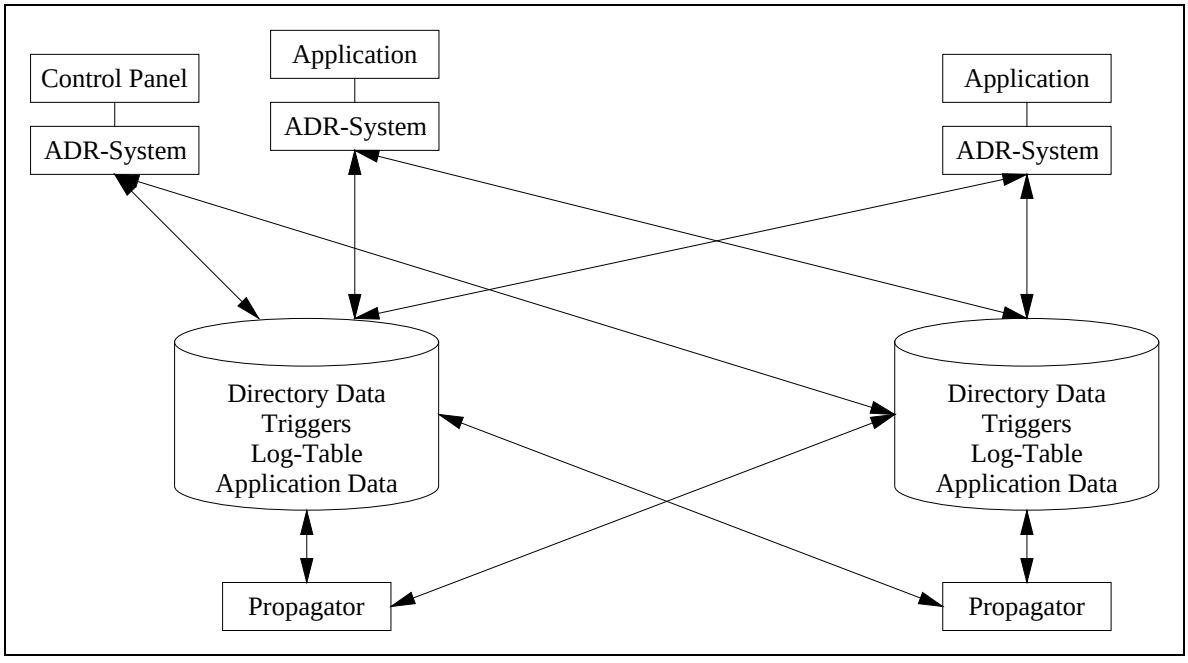

Figure 4: Overall system architecture

The databases hold the application data as well as meta-data, triggers and log-tables which are used to perform transaction processing and replica management according to the ADR formalism. So-called propagators at each database site are in charge of executing reproduction transactions correctly; they are independent from the application and its data. (Reproduction transaction are used to update secondary copies and are defined in section 3). The database administrator can use a control panel to change the replication schema (e.g. number and placement of secondary copies) or even to extend the relational database schema, without interrupting the current database activities or application programs.

2.4 Related Work

While basic theoretical results about replication have been known since the early 1980's [Davidson et al. 85; Skeen, Stonebraker 83], replication management has received renewed attention in the past few years, as commercial distributed database products are maturing in the market. Nevertheless, there is still a gap between the sophisticated techniques proposed in the literature and those employed in practice.

Surveys of replication management in databases can be found in [Abbott, Garcia-Molina 87], [Ceri et al. 91], [Chen, Pu 92], [Poledna 94] and [Beuter, Dadam 96]. In general, there are two groups of replication algorithms. The first group intends to preserve the classical consistency properties of the ACID concept [Härder, Reuter 83] through synchronous replication. The second group includes asynchronous techniques that allow for higher performance but usually cannot guarantee the ACID properties.

2.4.1 Replica control with full consistency

The classical *Read-One-Write-All (ROWA)* protocol [Bernstein, Goodmann 94] allows read operations to use any of the replicas while write operations have to be carried out synchronously on all replicas of a logical data objects. ROWA preserves full consistency because write operations lock all replicas and terminate with a commit protocol assuring the ACID properties. Due to its simplicity ROWA is implemented (along with two-phase commit (2PC)) in many commercial distributed DBMS, especially by software of vendors for open systems like the Oracle DBMS [Oracle 93]. Realizations based on transaction monitors (e.g. CICS, Tuxedo or DEC-ACMS [Gray, Reuter 93]) must use a standardized (X/Open) interface to achieve a two phase commit and often implement ROWA as well. However, the ROWA protocol slows down write operations and decreases write availability in case of communication or site failure. Furthermore, locking all replicas for write operations also decreases read availability. These performance drawbacks are not acceptable for many applications.

The *primary site* approach [Stonebraker 79] designates a distinguished site in the distributed database system to be the coordinator for all database items. All locks are kept at that site and all requests for locking or unlocking are processed by that site. This approach is a simple extension of a centralized locking strategy and hence easy to implement. The main disadvantages of the primary site solution is that the central site is a potential bottleneck regarding performance, availability and reliability, as each transaction requires communication with the primary site. Still, the primary site approach is used in hardware-oriented systems like Tandems RDF (*Remote Duplicate Database Facility*) for NonStop SQL and IBM's XRF (*Extended Recovery Facility*) [Tandem 96], [King et al. 91] and several telecom databases relying on high-performance computing equipment.

Beginning with [Gifford 79], *voting algorithms* of various complexity have been developed and also preserve full consistency. They carry out write operations on a subset of the replicas only and force read operations to read not only one but a certain number of replicas such that the most up to date

value can always be obtained. Thus, increased write availability is achieved at the expense of decreased read availability. However, it is the algorithms' complexity and uncertain performance which inhibited any implementation in commercial database systems so far [Liu et al. 95].

2.4.2 Replica control with relaxed consistency

The second group of algorithms intends to substantially improve performance through relaxation of the classical ACID requirements. The *ROWA-Available* approach (implemented e.g. in the VERSANT Replication System [Shyy et al. 1995]) is derived from the ROWA protocol by allowing write operations to modify only those replicas which are currently available. Thus, *ROWA-Available* buys write availability for relaxed consistency and not for reduced read availability. *ROWA-Available* can deal with site failures but not with communication failures.

Quasi-Copies [Alonso et al. 88] and epsilon-serializability [Pu 91] also trade consistency for performance and make the system converge asymptotically towards a consistent state. Similar to voting strategies, it is the complexity and missing transparency for the application which has prevented database vendors from implementing these techniques in their commercial products. The *SYBASE Replication Server* [Sybase 94] follows the idea to manage replicas asynchronously based on a *primary/secondary copy approach* such that consistency violations can be detected but not automatically resolved. It is then left to the application program or the database administrator to deal with the inconsistencies „manually". This technique is quite simple and allows high performance, as the bottleneck problems of the primary site approach can be distributed over several databases. However, it can happen that applications read secondary copies which represent a database state which never existed among the primary copies, i.e. serializability is violated.

[Gray et al. 96] discuss eager (synchronous) and lazy (asynchronous, delayed) propagation of updates and argue through an analytical model that only the (lazy) primary copy approach is suited to reduce the problems of system scalability. They show that, with synchronous propagation of updates, the deadlock and reconciliation rate would grow cubic with the number of nodes, number of transactions and transaction size.

Though independently developed [Gallersdörfer, Nicola 95], the ADR approach discussed in this paper also follows the (lazy) primary/secondary copy approach. Like the SYBASE replication server, ADR allows high performance transaction processing on replicated data but relaxes copy coherence only up to *defined leve*ls: therefore, as shown in section 3, applications can not access inconsistent database states.

In *Mariposa,* the management of replicated data is based on an economic framework [Sidell et al. 96]. Sites sell and buy copies of fragments from each other and generate revenue by query processing. Similar to ADR, Mariposa supports bounded *temporal* divergence between replicas. But unlike ADR, update propagation is done in a *non*-transactional fashion because more than one copy of a data item may be allowed to be written. The resulting conflicts then have to be resolved through a rule-based resolution system of considerable complexity. In ADR, only a primary copy may be written and (read/read or read/write) conflicts are not resolved but avoided through serializability. Thus, decoupled updates on secondary copies are not yet addressed in ADR, although they are useful in mobile computing applications.

An unconventional class of replica management protocols are epidemic algorithms. The basic idea is that updates will eventually reach all the replicas of a logical data item, similar to an infectious disease among the individuals of a population [Demers et al. 91], [Downing et al. 90]. The overhead of typical epidemic algorithms grows linearly with the number of physical data items. [Rabinovich

et al. 96] propose an improved epidemic algorithm which bounds the overhead to grow linearly with the number of data items that actually must be copied during propagation. Epidemic algorithms are useful for very large and heterogeneous distributed systems such as the Internet where the replicas of a logical data item may be spread over hundreds or even thousands of sites so that replicas will eventually (but not rapidly) converge towards a mutual consistent state [Demers et al. 91]. ADR is not designed for use in such widely federated information systems; it offers much more precise bounds on the divergence between replicas, but will work well only for a moderate number of replicas (less than 100).

Not all database approaches to telecommunications management problems are using distributed database technology at all. For example, the ClustRa main memory parallel database [Hvasshovd et al. 95] is based on a shared-nothing approach, ATM inter-node communication, and hot stand-by secondary copies to achieve high availability. A major difference to the ADR system is that ClustRa is expensively being built up from scratch while ADR is built on top of inexpensive commercial database technology. Other central main memory oriented telecom databases include the *Dali* system from AT&T Bell Labs [Jagadish et al. 94], Hewlett-Packard's *Smallbase* [Heytens 94] and Nokia's *TDMS* [Tikkanen 93]. These databases are highly specialized to specific telecommunication systems while ADR is sufficiently flexible to support a variety of applications.

3 Formal Properties and Implementation

The formal description of the ADR system has the main result that, given certain properties of the implementation, all operations in an ADR-based replication manager are conflict serializable, with a defined degree of aging.

3.1 Formal Definition of ADR

Let P be a site holding a primary copy x, and S a site holding a secondary copy x'. Considering x, we can call P a primary site and S a secondary site. The local concurrency control at each site ensures that the scheduling of local transaction[1] at P fulfills conflict serializability; thus, the local schedule corresponds to an acyclic conflict graph. Let T be the acyclic sub-graph which represents the projection of the local schedule on the committed transactions which accessed x.

A *reproduction transaction* (RPT) is a transaction which propagates the after image of x created by *one* of the transactions in T to the secondary copy x' at S. Practically, a RPT simply updates x' with the current value of x.

A *reproduction function* (RF) takes T as its argument and generates reproduction transactions. In particular, the RF defines for which of the transactions in T a RPT is generated. A RF implies *relaxed coherency* if it does not generate RPT's for *all* transactions in T. Since T only contains transactions that accessed x, a reproduction function with relaxed coherency means that the secondary copy x' is not necessarily updated every time the primary copy x has been changed.

A RF is *correct* if the order in which the reproduction transactions are generated agrees with T, i.e. if RPT_i denotes the reproduction transaction for a transaction $t_i \in T$, then the order of the RPT's corresponds to one of the partial orderings implied by T.

A set of reproduction transactions is *executed correctly*, if the local conflict graph at the secondary site S agrees with T, i.e. the reproduction transactions on x' are executed in the order in which they have been generated by the correct reproduction function. This means, secondary copies are refreshed in a monotone way.

[1] Local transaction are transactions that access only data items at the local site.

Theorem [Gallersdörfer 1997]: *If each transaction has at most one write step, and if reproduction transactions are generated by a correct reproduction function, and if these reproduction transactions are executed correctly, then every read-step at a secondary copy reads a state which once was a valid state of the primary copy*.

Proof: The formal proof is given in the Appendix. The basic argument is that the local schedule at the primary copy is conflict serializable which manifests in the acyclic conflict graph T. The correct generation and execution of reproduction transactions implies that the partial ordering of T is maintained when updating the secondary copy (i.e. neither network delay nor local scheduling can disturb the order of RPT execution). This is because read-steps and reproduction transactions at the secondary copy are executed in a conflict serializable schedule guaranteed by local concurrency control. Nevertheless it is crucial that the write operations in a write-step do not depend on any value read in any of the read-steps. If a write-operation needs to depend on a read-operation, both the read and the write have to be executed within the write-step on the primary copy. Otherwise, the value of the primary copy could depend on an aged secondary copy such that external consistency (and subsequently (after update propagation) internal consistency) was violated.

The conflict serializability of the ADR database follows from this theorem; more formal details, definitions and proofs can be found the Appendix and in [Gallersdörfer 97].

Corollary: Each individual read-step reads the same data as in a conflict serializable schedule, if the reproduction function is correct, every write step schedule is conflict serializable, and every schedule of reproduction transactions and read steps is correctly executed.

The implementation of ADR must ensure that all the formal requirements of the corollary are satisfied. In addition, it must also monitor that the user-defined bound on database coherency [Gallersdörfer, Nicola 95] is preserved. More precisely, the main requirements posed by the formalization on the implementation of the ADR system are:

(1) *Management of primary and secondary copies*: A compact and efficient representation of meta-data about the placement of primary and secondary copies is required for processing of user transactions and propagating updates to secondary copies.

(2) *Correct generation of reproduction transactions*: The system must be able to detect changes of primary copies and has to generate reproduction transactions in the correct order.

(3) *Correct execution of the reproduction transactions*: Secondary copies have to be updated independently at the different database sites under local concurrency control and reproduction transactions may not overtake each other.

It turned out that such a realization is not straightforward. The following subsections describe how these three problems have been solved within the ADR architecture discussed in section 2.3.

3.2 Management of primary and secondary partitions

Partitions contain records from various tables. We use horizontal fragmentation only, to avoid repetition of primary keys and to simplify the implementation of a trigger mechanism which is fundamental to ADR (see below). So-called *partition keys* are used to define which records are grouped to form a partition. A partition key is the minimal set of attributes of a relation which uniquely identify the partition a record belongs to. Partition keys are defined by triggers; they detect changes of primary copies, and compute for a given record of any table the name of its partition.

Each database site maintains a partition directory that holds information about which primary and secondary copies are available at which sites, in particular which copies are available *locally*. This directory data is rarely changed but a read access is necessary for every execution of a user submitted transaction, for routing transaction steps to sites holding appropriate partitions. Therefore maximum performance is achieved by replicating the complete partition directory to all sites.

The partition directories are changed only when partitions are created, moved, deleted or replicated. The corresponding maintenance of the partition directories is very efficient because it is managed in the same way as user data. The directories are common database tables and the directory records concerning a certain partition are defined to be part of the partition itself. Thus, every partition contains a number of application data records and 1+*s* partition directory records where *s* is the number of secondary copies of the partition.

When a partition is replicated using the `replicate` command provided by the ADR module the *scope of the replication* can be specified to be either `complete` or `directory`. In case of *complete replication* a secondary copy of the complete partition is created. In case of *directory replication* only the directory records of the partition are replicated. A directory record describing a secondary copy contains a field `scope` which tells whether the secondary copy is a `complete` or a `directory` replica. Now every primary partition is replicated in the `directory` mode to all sites which results in a fully replicated partition directory. „Real“ secondary copies are created in the `complete` mode for selected partitions and sites only.

Whenever a partition is created, moved, deleted or replicated, a corresponding change in the local partition directory is detected and propagated as if it was user data. This ensures that any local directory is always a correct description of the application data placement and that the evolution of the replication schema can take place while the systems remains in full operation.

For primary copies, a corresponding record in the directory only holds the name of the partition. A directory record for secondary copies also contains the scope of its replication, the name of the site of its primary copy as well as two fields named `RefreshTime` and `Aged` which are used for update propagation. `RefreshTime` specifies a duration *m* which declares that the secondary copy has to be updated every *m* time units. In case scheduled updates of the secondary copy fail (e.g. due to a site or communication failure) `Aged` signals that the coherency requirement for the secondary copy is violated and shows the number of unsuccessful consecutive attempts to update. During normal operation `Aged` is zero. However, the partition directories are not only used for update propagation but also for the general processing of user submitted transactions. Both aspects are described in more detail in the next sections.

3.3 Correct generation and execution of reproduction transactions

Another problem is the detection of changes of primary copies (INSERT, UPDATE, DELETE). The internal database log which records all modifications is typically not available for examination. But even if it was, we would refrain from using it to keep the ADR system independent from a certain log format or a certain database vendor. Instead, we attach triggers to every application data table. The relational DBMS provides the triggers with the values the changed records had prior and after the modification so that the triggers can insert the records’ primary keys, their partition names and new values (if any) into a log *table*. Furthermore, each entry of the log table has a flag to indicate whether the registered modification was an insert or delete. Updates are logged as an insert followed by a delete because updating a record might move it from a partition P1 to a partition P2 which requires to insert it into copies of P2 and to delete it from the copies of P1, which may reside at

different sites. A trigger to register the deletion of a primary record in the table DB_Service is implemented as shown below:

```
CREATE TRIGGER DB_TRG_DEL_Service
   ON DB_Service
   FOR DELETE
   AS
   BEGIN
      DECLARE @pname    DB_TYPE_part_name
      SELECT  @pname  = "S."+service+subscriber FROM deleted
      IF (DB_is_primary(@pname))
        INSERT DB_Log_Table
            SELECT @pname, "DB_Service", service, "Del", NULL, 1
            FROM deleted
   END
```

The triggers are as short as possible because they are executed within the context of an original user transactions which should not be slowed down. Performance measurements in our implementation showed that the triggers incur only a minor overhead which does not significantly affect the overall application performance. It is important to note that update propagation is not launched by the triggers themselves in order to preserve easy recovery: if the trigger's transactional context was aborted after the trigger sent a reproduction transaction to a remote database, the local database undoes the trigger, and the transaction it fired on, automatically but not the remote RPT, which creates a difficult distributed recovery and atomicity problem. Hence, update propagation is done in a completely decoupled fashion.

An important design decision was whether the primary site should initiate to *push* the RPT's to the secondary site or whether the secondary site should initiate to *pull* them from the primary site. With a *push* strategy it is extremely difficult to detect that a secondary copy is aged (i.e. its coherency condition is violated) because a secondary site cannot decide whether a lack of incoming RPT's is due to a site or communication failure or simply because the primary copy did not get modified for a while. Therefore the generation and execution of RPT's is initiated as a *pull* by propagators at the secondary sites (cf. Figure 4).

The local partition directories provide information about which local secondary copies have to be updated at which times and where the appropriate primary copies are located. Using this information, the propagators ensure timely refreshment of secondary copies: If a partition directory says that a local secondary partition x' is to be updated every *m* time units and that the corresponding primary copy x resides at a site P, then every *m* time units the propagator contacts site P and reads those records from the modification log table at P which represent the changes that happened to records in x since the last update of x'. These records from the log table provide sufficient information for the propagator to launch RPT's locally to update x'. In practice, the propagator simply calls precompiled SQL transactions (*stored procedures*) and passes the information read from the remote log table on to them as parameters. In case the propagator fails to read the log table at P (site or communication failure) it marks the local secondary copy as `Aged`. It is then left to the application to decide whether the stale copy will do or not.

To realize relaxed coherency, the log table records are timestamped. If now multiple changes happen to a primary record, only the latest state of the primary record is used to refresh the secondary copy. This reduces the number of RPT's and leads to an overall performance improvement (see section 4).

The implementation of forming and executing RPT's is correct regarding the definition in section 3: RPT's are executed in the order of their timestamps which have been generated by triggers within the context of the original transactions and thus reflect their commit order. To ensure that RPT's are serialized in commit order, the DBMS responsible for the management of a secondary copy needs to support strict two-phase locking.

4 Evaluation of ADR

The evaluation of ADR presented in this section highlights two of the main features of ADR. Firstly, it quantifies how the controlled relaxation of coherency in ADR can be exploited to fulfill given performance requirements, based on an analytical model which has been validated against measurements on the implementation. Secondly, it demonstrates that ADR is sufficiently flexible to manage secondary copies which reside either in conventional relational database tables or in main memory structures if response time requirements dictate to do so.

ADR has been used in two real-world applications to allow an authentic and meaningful evaluation. One is a distributed database for *Intelligent Network* (IN) telephone services; the other is real-time data support for mobile phones in a city-wide DECT setting. The main performance goals of the IN application are high throughput and scalability while providing sufficiently low response times [Gallersdörfer et al. 94]. In the city-wide DECT application the most critical performance requirement is extremely short response times. Experiences with these two applications have shown that ADR is indeed suitable to allow for high throughput and scalability or for very short response times respectively by relaxing coherency in a controlled manner.

In section 4.1 we sketch a slightly revised version of our queueing model for performance estimations originally presented in [Gallersdörfer, Nicola 95], sections 4.2 and 4.3 then present the experiences with ADR in the Intelligent Network and city-wide DECT application.

4.1 The queueing model

In [Nicola 95], [Gallersdörfer, Nicola 95], we developed an analytical queueing model to evaluate performance improvements gained through relaxed coherency in general and adapted it to assess the performance of ADR in the IN context. For this paper to be self-contained, this section contains a summary of the analytical queueing model used in sections 4.2 and 4.3. Further details including justification for the modeling assumptions and derivation of the performance values can be found in [Gallersdörfer, Nicola 95] where also the calibration and validation of the model in the IN context is described.

4.1.1 Parameters and modeling assumptions

In our model a replicated database consists of n identical *local databases* (or *sites*). Identical means that the sites use the same database hardware and software, hold the same amount of data, receive the same workload, etc. This distributed database is modeled as an open queueing network. Its nodes are identical $M/H_2/1$-systems characterizing the local databases.

We model the arrival of queries and updates to each local database by Poisson streams with parameters λ_q and λ_u respectively. The percentage of queries in the overall workload is denoted by the parameter $a_q \in [0;1]$. The users submit their transactions to a local site, which may need to forward the execution to another site due to a lack of appropriate local data. Additionally, our model is based on the assumption that updates are executed according to the *primary copy approach*. We assume a good design in the sense that each transaction accesses data items of only one database

because transactions are expected to reference logically dependent data items which should be grouped together. The *quality of data distribution* is modeled by the probability ($loc \in [0;1]$) that a transaction can be executed at the local site. With probability $1 - loc$ a transactions has to be forwarded to one of the remaining sites, each of which being chosen with equal probability.

The controlled degree of relaxed coherency is modeled by the *coherency index* $k \in [0;1]$. Small values of k express high relaxation and expected low costs for update propagation. A value of $k = 0$ models suspended updated propagation, i.e. values of replicated data objects age unlimited. For $k = 1$ all updates have to be propagated immediately which does not imply synchronous update.

Many models of replicated databases assume *full* replication. In contrast, we believe that *partial* replication is necessary to achieve high performance. Therefore we model the degree of replication by the parameter $r \in [0;1]$ describing the percentage of logical data items that are fully replicated across the sites. That means that *if* a data item is replicated, a copy exists at each site. Updates executed at a local database therefore have to be propagated to all other sites with probability r. Considering relaxed coherency, the *probability of propagation* decreases to $k \cdot r$. The model of replication used here is a 1-dimensional model of partial replication because it varies the fraction of replicated data items but keeps the number of their copies fixed to the number of sites n. This is state of the art in performance models for replicated databases. In follow-up work which will not be further discussed here, we propose a more advanced 2-dimensional model of replication [Nicola, Jarke 99].

The *quality of replication* depends on the preference of queries and updates to access replicated data. These preferences are modeled by the parameters $qr \in [0;1/r]$ and $ur \in [0;1/r]$ respectively. Here, a value of $qr = 1/r$ ($ur = 1/r$) describes that queries (updates) are accessing replicated data only; meaning an optimal (unskillful) replication schema. The value 0 expresses the opposite extreme while the value 1 describes no preferences. Considering the parameters *loc*, *qr* and *ur*, we clearly refrain from assuming uniformly distributed access to data objects across the database, as a major difference to most models proposed in literature. The *probability of propagation* now amounts to $r \cdot k \cdot ur$.

In order to distinguish between updates and queries we model the query (update) service time to be exponentially distributed with mean t_q (t_u) seconds. This leads to the two phase hyperexponential distribution of the service time for the combined flow [Kleinrock 75]. The communication network is assumed to affect the performance by introducing a constant delay in every intersite communication, which takes $t_n^{message}$ seconds for short messages and t_n^{data} seconds for transmitting data (e.g. query results).

Parameter	Description
n	Number of sites (system size)
a_q	Percentage of read-only transactions (queries)
λ_q	Local arrival rate at each site: queries/sec
λ_u	Local arrival rate at each site: updates/sec
loc	Quality of data distribution
r	Degree of replication
qr	Preference of queries reading replicated data
ur	Preference of updates changing replicated data
k	Coherency index
$t_n^{message}$	Communication delay for short messages
t_n^{data}	Communication delay for transmissions of data
t_q	Average query service time
t_u	Average update service time

Table 1: Model parameters.

4.1.2 Arrival rates and response time

The probability that submitted queries (updates) can be executed at the local database is denoted as ℓ_q (ℓ_u respectively) and results in

$$\ell_q = loc + (1 - loc) \cdot r \cdot qr \qquad \text{and} \qquad \ell_u = loc$$

because *loc* expresses the preference of accessing original local data and the second term reflects the local read availability introduced by replication. Note that replication does not increase the write availability because of the primary copy approach.

The overall rate of queries to be executed at a local database (λ_q^{total}) includes queries submitted directly by users as well as additional queries forwarded from other sites:

$$\lambda_q^{total} = \ell_q \cdot \lambda_q + (n-1) \cdot (1 - \ell_q) \cdot \lambda_q \cdot \frac{1}{n-1} = \ell_q \cdot \lambda_q + (1 - \ell_q) \cdot \lambda_q = \lambda_q$$

Considering the identical behavior of sites, $\lambda_q^{total} = \lambda_q$ is not much of a surprise: every site receives just as many queries as it forwards to other sites due to a lack of appropriate local data. For updates we similarly derive

$$\lambda_u^{total} = \ell_u \cdot \lambda_u + (n-1) \cdot (1 - \ell_u) \cdot \lambda_u \cdot \frac{1}{n-1} + (n-1) \cdot r \cdot k \cdot ur \cdot \lambda_u = (1 + (n-1) \cdot r \cdot k \cdot ur) \cdot \lambda_u$$

In addition to the rate of locally submitted updates (λ_u) the amount of propagated updates has to be included: An arbitrary update must be propagated with probability $r \cdot k \cdot ur$ at each of the n-1 remaining sites.

Since the service time of the combined stream is hyperexponentially distributed and each node acts like a $M/H_2/1/FCFS$ system the *average waiting time* $\overline{W}$ at a local database can be derived using the *Pollaczek-Khinchin formula* [Kleinrock 75]:

$$\overline{W} = \frac{\lambda_q^{total} t_q^2 + \lambda_u^{total} t_u^2}{1 - \lambda_q^{total} t_q - \lambda_u^{total} t_u}$$

Using this result we can determine the *average response time for queries* which amounts to

$$\overline{R}_q = \ell_q \cdot (\overline{W} + t_q) + (1 - \ell_q) \cdot (t_n^{message} + \overline{W} + t_q + t_n^{data})$$

The first term of $\overline{R}_q$ corresponds to queries that can be answered locally and the second term covers the case that queries have to be forwarded to another site, taking $t_n^{message}$ seconds and requiring the results to be sent back, taking t_n^{data} seconds. Similarly, the *average response time for updates* is

$$\overline{R}_u = \ell_u \cdot (\overline{W} + t_u) + (1 - \ell_u) \cdot (t_n^{message} + \overline{W} + t_u + t_n^{message})$$

The overall average response time can defined as $\overline{R} = a_q \cdot \overline{R}_q + (1 - a_q) \cdot \overline{R}_u$.

In steady state the number of arriving transactions equals the number of departing transactions, so that the *throughput* of the distributed database equals the arrival rate. However, the overall throughput of the system is bounded by the capacity of the local sites, i.e. the utilization of the sites cannot exceed 100%. Therefore we derive the *maximum throughput D* by solving the equation $\lambda_q^{total} t_q + \lambda_u^{total} t_u = 1$ for λ^{global} ($\lambda^{global} = n \cdot \lambda_u + n \cdot \lambda_q$) which results in

$$D = \left(\frac{a_q}{n} \cdot t_q + (1 + (n-1) \cdot r \cdot k \cdot ur) \cdot \frac{(1 - a_q)}{n} \cdot t_u \right)^{-1}$$

4.2 Integrating Database Design and Operation for Intelligent Network Services

Our research was initiated by the need to provide database support for an Intelligent Network design and operations environment developed by Philips Labratories. The IN is an architectural concept for telecommunication networks that enables network operators as well as independent service providers to swiftly introduce new services such as free-phone, virtual private network, televoting, etc. into existing networks. Furthermore, these services should be made sufficiently flexible so that after deployment, service subscribers can tailor them to their requirements.

The main idea of the IN concept is the separation of switching functionality from service control. To achieve a high degree of flexibility, the service logic is realized by software modules called *service logic programs* (SLPs) which can be customized with subscriber specific data. Figure 5 shows the structure of the Intelligent Network as it is defined by the IN standards ITU-T (former CCITT) CS1 and AIN.

A *Service Switching Point* (SSP) recognizes calls from an end user phone to a service which requires support by a *Service Control Point* (SCP) and sends an instruction request to the SCP. A SCP retrieves the corresponding *Service Logic Program* (SLP) and service data from the database, evaluates it and sends a response back to the SSP. The *Service Creation Environment* (SCE) is used for creation and testing of new services which are then transferred via the SMS to the SCP. The *Service Management System* (SMS) is needed for downloading service logic programs and service data as well as for other management activities such as billing and statistics.

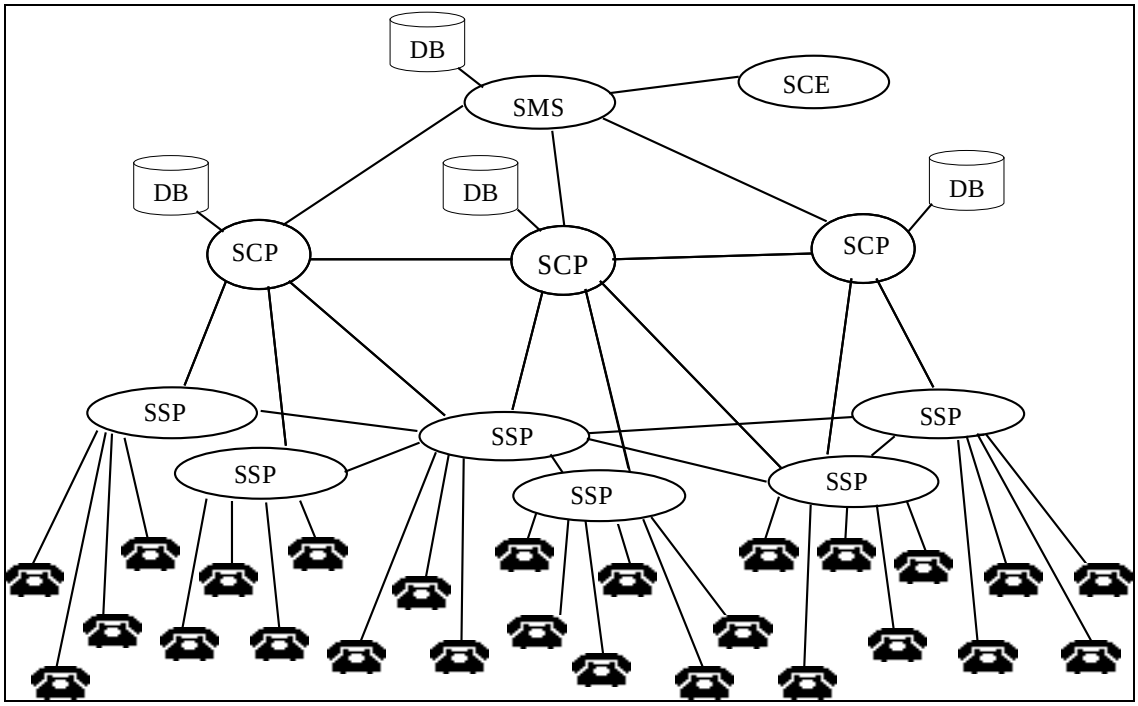

Figure 5: The Intelligent Network Architecture

Such an IN system has to handle large amounts of data (SLPs, subscriber specific data, management information). Several IN vendors are using a large central mainframe database system to provide consistent data support for all SLPs running at the same time. However, such systems are not only very expensive but also a potential bottleneck regarding availability and scalability. Furthermore, every data request requires communication with the central site. As telecommunication systems are of highly parallel nature, a large scale distributed database system composed of commodity hardware can be a more natural and less expensive solution [DeWitt, Gray 92]. Hence, a main design goal of our ADR-based implementation was to provide high performance and highly scaleable replica management on top of standard database hardware and software. The implementation of ADR in the IN database followed the outline in section 3 and is elaborated in detail in [Gallersdörfer et al. 96].

Using the queuing model from section 4.1, one of the most important performance results for the IN application is given in Figure 6. It shows the maximum throughput as a function of the number of sites n where the coherency index is taking the values 0, 0.1, 0.25, 0.5, 0.75 and 1 (from top to bottom). If the percentage of updates is not negligible (like 10% in Figure 6), throughput does not increase linearly with the number of sites due to update propagation overhead (when $k > 0$). However, the graphs for $k < 1$ indicate that relaxed coherency may improve scalability towards ideal linearity [DeWitt, Gray 92]. Figure 6 also shows that for a given number of sites throughput can be increased by relaxing coherency, and the larger the system the greater the gain.

The analytical model has been validated through measurements of ADR in the IN implementation (section 5 in [Gallersdörfer, Nicola 95]). It has shown that ADR is a very suitable concept for the transaction processing and replica management which is required to provide distributed database support for an intelligent network. In particular, the coherency trade-offs in ADR allow for sufficient throughput and scalability as demanded by a large scale distributed telecommunication system.

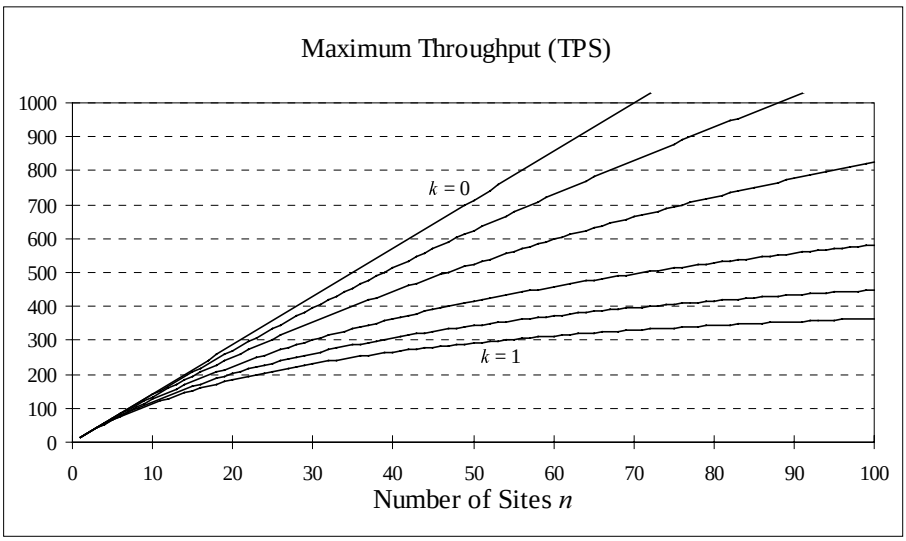

Figure 6: Maximum throughput vs. system size for 90% read-only transactions.

4.3 Managing Mobility Data in the City-Wide DECT

The *City-Wide DECT* is a mobile, wireless telephone network using very small cells[2]. In the Philips implementation, so called Mobility Managers (MM) provide the system with profile and location information about the users. This information has to be administrated in a distributed (replicated) database due to the distributed nature of the overall system. As for the intelligent network, we believe that the MM components and their databases should be realized on commodity hardware available in the mass market for cost reasons and independence of a special hardware manufacturer.

In our implementation, a reference copy of the complete information (location and profile) is placed as a primary copy in a conventional relational database management system. This sub-component provides reliability by means of classical storage on persistent media (hard disks). In order to achieve high availability of the primary data we used concepts directly supported by the RDMBS system, like RAID (redundant arrays of independent disks [Gibson 92]). For recovery reasons, the data and the log of the database were placed on separate disks. Furthermore, the RDBMS directly supports the mirroring of data-disks and log-disks. Our experiences show that these high availability concepts did not slow down the RDBMS machine. In order to increase availability we introduced multiple hot-standby machines. The redundant secondary copies are driven through ADR concepts, i.e. relaxed consistency and asynchronous propagation of updates.

In contrast to the IN application, the most critical performance requirement of the City-Wide DECT application is a very short *response time*. A mobile phone user expects a dial tone less than one second after she lifts the receiver. Analyzing the tasks to be performed during this second (i.e. „attach“ to the network) led to the result that at most 10 ms can be spared for database access. The performance evaluation of the IN application indicated that such a response time can hardly be achieved with today’s commodity database and disk technology. Indeed, additional measurements with a Sybase SQL Server proved that even a single centralized database server is not able to fulfill such a tough response time requirement. We saturated the database with a mix of write- and read-only transaction, gradually increased the share of write transactions from 0% to 100% and measured the response time of each transaction. The histogram in Figure 7 depicts the number of transactions which completed within a certain response time limit for a given read/write mix. Even for 0% write transactions one third of the transactions had a response time of more than 100 ms.

[2] DECT stands for Digital Enhanced Cordless Telecommunications.

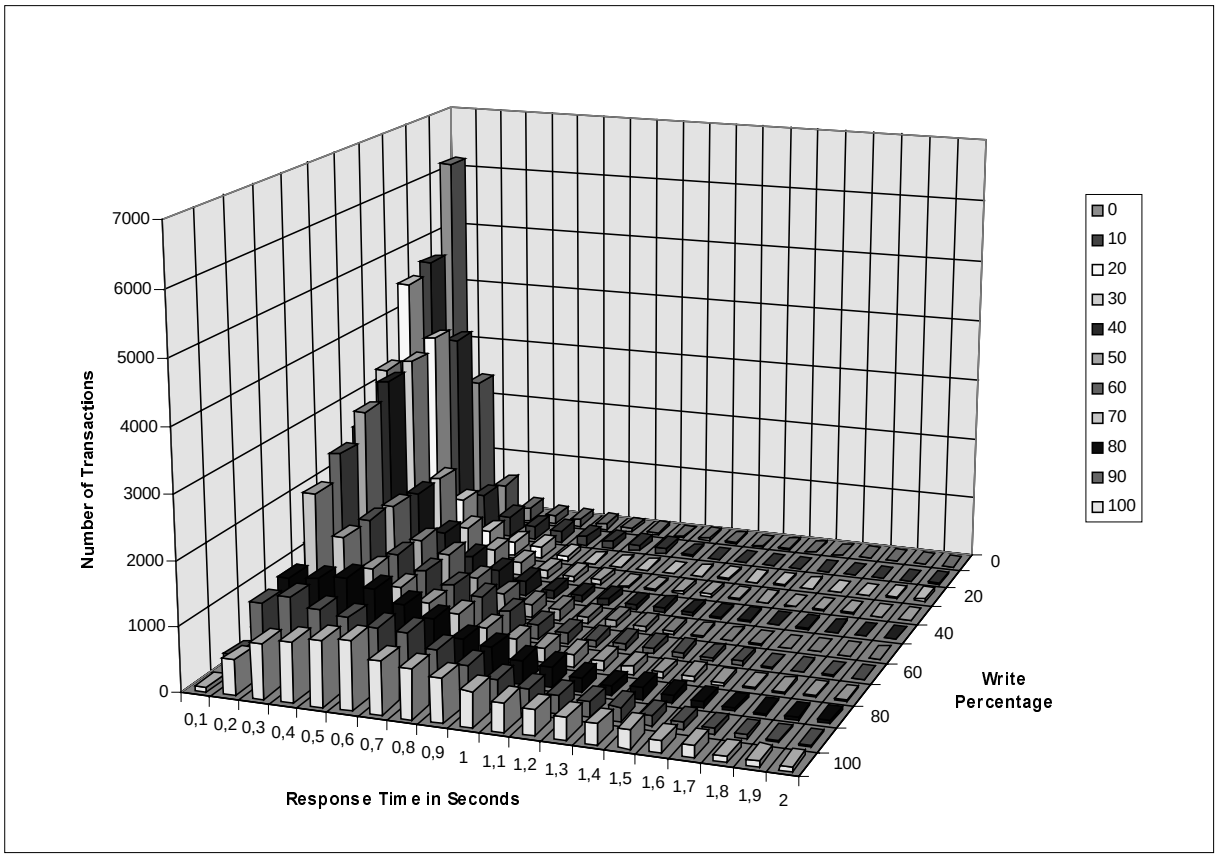

Figure 7: Response times of a centralized database server

Interestingly, the ADR approach works equally well if secondary copies are kept in main memory databases or caches; such caches were therefore placed in each MM. This allows to access the location data without disk I/O or remote access to the primary copy. The most difficult problem that remained was to propagate the location updates that occur in a distributed manner. We relaxed the coherency of the secondary copy in a time oriented manner through delayed propagation of updates. The City-Wide DECT is meant to provide seamless service for users at up to walking speed; thus, location information can be tolerated to age up to 10 seconds due to overlapping cells. Thus, updates at the primary copy are collected for at most 10 seconds and then written to the secondary copies in a single transaction. As shown below, our analytical model predicted that this will achieve a response time below 10 ms while guaranteeing that the MM process never accesses data older than 10 seconds. Extensive measurements on the implementation confirmed these results.

The City-Wide DECT application can be modeled by setting the number of sites n to 10 and the overall transaction arrival rate to 100 TPS, where only 50% are assumed to be read-only transactions, because decreasing cell sizes and high traffic rates will lead to a very high update rate for location information in mobile telephone networks [Lo, Wolff 93]. This means the distributed system has to execute 50 updates per second leading to 5 updates per seconds per site. Thus, within the 10 seconds of update propagation delay a number of 50 updates are accumulated into 1 update transaction which is forwarded to the secondary copies. This yields a coherency index of $k = 0.02$.

Figure 8 shows the expected average transaction response time as a function of the degree of replication for different levels of coherency requirements. The case $k = 1$ represents asynchronous but immediate propagation of updates. In this situation the response time can be reduced remarkably by replicating about 30% of the data. This leads to increased local access, while a higher degree of replication rapidly saturates the local databases with propagated updates ($r > 0.4$). The graph for

$k = 1$ shows that even with an optimal degree of replication asynchronous but immediate update propagation would prevent the database from satisfying the response time requirement so that a relaxation of coherency is necessary. The curve for $k = 0.5$ represents the case of refreshing the secondary copies after every second update and $k = 0.1$ means to delay update propagation for intervals of 2 seconds. However, neither strategy is suitable to decrease response time below 10 ms. The allowed delay of 10 seconds ($k = 0.02$) has to be fully exploited to reduce the load of processing reproduction transactions far enough such that full replication becomes affordable and response time drops below 10 ms. In order to verify this result we carried out measurements in our ADR implementation for the City-Wide DECT application. We generated about a million typical City-Wide DECT transactions and found that in the case of full replication and 10 seconds update propagation delay, indeed 99% of the transactions had a response time of less than 10 ms.

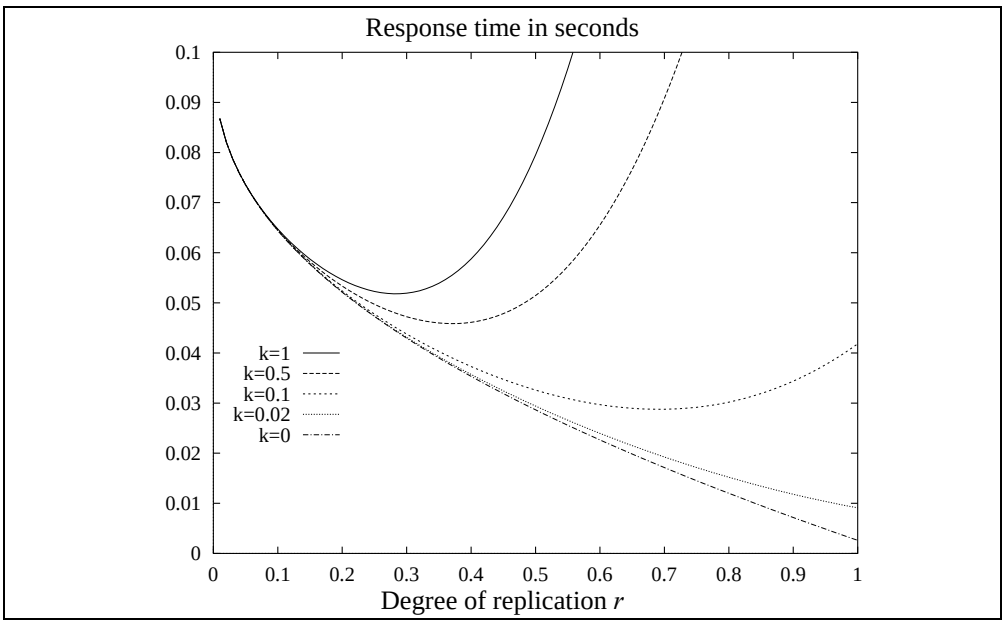

Figure 8: Response time results for the City Wide DECT application

5 Summary and Conclusions

ADR is a correct, robust and efficient management system for replicated data on top of standard relational database technology. ADR is based on the primary copy approach and delayed asynchronous update propagation, but still preserves defined levels of consistency through application oriented data partitioning and executing the transactions' read- and write steps on partitions of sufficient consistency. Thus, we claim that ADR withstands the criticism in [Goldring 95] that most implementations of asynchronous replica control provide insufficient consistency properties. We described how ADR can be implemented as a simple and lean system on top of existing database technology.

The intelligent network and City-Wide DECT are two demanding and sophisticated applications in the rapidly expanding field of telecommunications where ADR has proven to be an appropriate solution for the required tradeoff between database performance and consistency. We expect that there are a number of other application domains in which ADR's controlled relaxation of coherence, combined with a group-oriented approach to distributed database design can be useful. For example, a group at INRIA is currently addressing the problem of incremental view refreshment in data

warehousing based on a variant of ADR which uses a slightly different replication and propagation policy [Pacitti, Simon 98].

In addition, a number of extensions to this work are currently underway in our own follow-up projects. Our application studies in the health-care and mobile computing sectors indicate the need for improving performance evaluation techniques for replicated database. In particular we focus on a balanced modeling and evaluation of both the communication as well as database issues in the distributed system. This allows for analyze more design options and broader bottleneck analysis [Nicola, Jarke 98], [Nicola, Jarke 99]. Another issue is the explicit consideration of decoupled computing modes, with a modified primary copy approach allowing for decoupled updates. This will require the integration of the secondary-copy handling strategies of ADR with conflict resolution strategies as, e.g., proposed in Mariposa.

Acknowledgments. This work was supported in part by Philips Research Laboratories, Aachen, by the Deutsche Forschungsgemeinschaft in its Special Doctoral Program "Informatics and Engineering" at RWTH Aachen, and by the Commission of the European Communities under ESPRIT Long Term Research project DWQ. Some of the results reported here have been protected by international patents held by Philips N.V. We wish to thank Karin Klabunde, Martin Elixmann, Ralf Nellessen, Axel Stolz and Marco Essmajor for their support and cooperation during this project.

References

[Abbott, Garcia-Molina 87] R. Abbott, H. Garcia-Molina: „*Reliable Distributed Database Management*", Proceedings of the IEEE, 75 (5), May 1987, pp. 601-620.

[Alonso et al. 88] Rafael Alonso, Daniel Barbara, Hector Garcia-Molina, Soraya Abad: „*Quasi-Copies: Efficient Data Sharing for Information Retrieval Systems*", Proceedings of the International Conference on Extending Data Base Technology (EDBT), pp. 443-468, 1988.

[Bernstein, Goodman 84] Philip. A. Bernstein, Nathan Goodman. „*An Algorithm for Concurrency Control and Recovery for Replicated Databases*." ACM Transactions on Database Systems, 9(4), 1984.

[Beuter, Dadam 96] T. Beuter, P. Dadam: „*Principles of replication control in distributed database systems*" (in German), Informatik Forschung und Technik, Vol. 11, No. 4, pp. 203-212, 1996.

[Ceri et al. 91] S. Ceri, M.A.H. Houtsma, A.M.Keller, P.Samarati: „*A Classification of Update Methods for Replicated Databases*", Technical Report STAN-CS-91-1392, Stanford University, 1991.

[Chen, Pu 92] Shu-Wie Chen, Calton Pu: „*A Structural Classification of Integrated Replica Control Mechanisms*", Technical Report CUCS-006-92, Columbia University New York, 1992.

[Davidson et al. 85] Susan B. Davidson, Hector Garcia-Molina und Dale Skeen. „*Consistency in Partitioned Networks*", ACM Computing Surveys, 17(3):341-370, September 1985.

[Demers et al. 91] Alan Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swineheart, D. Terry: „*Epidemic Algorithms for replicated database maintenance*", Technical Report, Xerox Palo Alto Research Center, CSL-89-1, January 1991.

[DeWitt, Gray 92] D. DeWitt, J. Gray: „*Parallel Database Systems: The Future of High Performance Database systems*", Communications of the ACM, Vol. 35, No. 6, June 1992, pages 85-98.

[Downing et al. 90] Alan R. Downing, Ira B. Greenberg, Jon M. Peha: „*Oscar: A system for weak-consistency replication*", Proceedings of the 1st Workshop on the Management of Replicated Data, Houston, November 1990, pp. 26-30.

[Gallersdörfer 97] Rainer Gallersdörfer: „*Replikationsmanagement in verteilten Informationssystemen*“, Ph.D. thesis (in German), RWTH Aachen, Informatik V, February 1997.

[Gallersdörfer et al. 96] R. Gallersdörfer, K. Klabunde, A. Stolz, M. Eßmajor: „*Intelligent Networks as a Data Intensive Appliction - Final Project Report*“, Technical Report AIB-96-14, ISSN 0935-3232, Technical University of Aachen, June 1996.

[Gallersdörfer et al. 94] Rainer Gallersdörfer, Matthias Jarke, Karin Klabunde: „*Intelligent Networks as a Data Intensive Appliction (INDIA)*“, Proceedings of the 1st International Conference on Applications of Databases, Vadstena, Sweden, June 1994.

[Gallersdörfer, Nicola 95] Rainer Gallersdörfer, Matthias Nicola: „*Improving Performance in Replicated Databases through Relaxed Coherency*“, Proceedings of the 21th International Conference on Very Large Database, pp. 445-456, September 1995.

[Gibson 92] Garth A. Gibson. „*Redundant Disk Arrays: Reliable, Parallel Secondary Storage*“, An ACM Distinguished Dissertation 1991. MIT Press, 1992.

[Gifford 79] D. K. Gifford. „*Weighted Voting for Replicated Data*“, ACM Symposium of Operating Systems Principles (SOSP), pp. 150-162, Pacific Grove CA, December 1979.

[Goldring 95] R. Goldring: „*Things every update replication customer should know*“, in SIGMOD Record, Vol. 24, No. 2, pp. 439-440, June 1995 or in InfoDB, Vol. 9, No. 2, April 1995.

[Gray et al. 96] Jim Gray, P. Helland, P. O’Neil, D. Shasha: „*The dangers of replication and a solution*“, SIGMOD Record, Vol. 25, No. 2, pp. 173-182, June 1996.

[Gray, Reuter 93] Jim Gray, Andreas Reuter: „*Transaction Processing Concepts*“, Morgan Kaufmann, 1993.

[Härder, Reuter 83] Theo Härder, Andreas Reuter: „*Principles of Transaction-Oriented Database Recovery*”, ACM Computing Surveys 15(4), pp. 287-317, 1983.

[Heytens 94] M. Heytens, S. Listgarten, M.A. Neimat, K. Wilkinson: „*Smallbase: A main memory DBMS for high performance applications*“, Hewlett-Packard Laboratories, March 1994.

[Hvasshovd et al. 95] S.O. Hvasshovd, O. Torbjornsen, S.E. Bratsberg, P. Holager: „*The ClustRa telecom database: high availability, high throughput, and real-time response*“, Proceedings of the 21st VLDB, pp. 469-477, September 1995.

[Jagadish et al. 94] H.V. Jagadish, D. Lieuwen, R. Rastogi, Avi Silberschatz: „*Dali: A high performance main memory storage manager*“, Proceedings of the 20th International Conference on Very Large Databases, pp. 48-59, September 1994.

[Kerboul 93] R. Kerboul, J.M. Pageot, V. Robin: „*Database Requirements for Intelligent Networks: How to customize mechanisms to implement policies*“, 4th Telecommunications Information Networking Architecture Workshop, Volume II, September 1993.

[King et al. 91] R.P. King, N. Halim, H. Garcia-Molina, Christos. A. Polyzois: „*Management of a Remote Backup Copy for Disaster Recovery*“, ACM Transactions on Database Systems, 16, 2, 1991.

[Kleinrock 75] Leonard Kleinrock: „*Queueing Systems, Volume I: Theory*“, John Wiley & Sons, 1975.

[Liu et al. 95] M.L. Liu, D. Agrawal, A. El Abbadi: „*The performance of replica control protocols in the presence of site failures*“, Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, pp. 470-477, October 1995.

[Lo, Wolff 93] C. N. Lo, R. S. Wolff. „*Estimated Network Database Transaction Volume to Support Wireless Personal Data Communications Applications*“, Proceedings ICC '93, Genf, May 1993.

[Nicola 95] Matthias Nicola: „*Analytical Performance Evaluation of Relaxed Coherency in Replicated Databases*“, Diploma thesis (in german), RWTH Aachen, Informatik V, June 1995.

[Nicola, Jarke 98] M. Nicola, M. Jarke: “*Design and Evaluation of Wireless Health Care Information Systems in Developing Countries*”, Proceedings of the IFIP 9.4 Conference on Implementation and Evaluation of Information Systems in Developing Countries, February 1998.

[Nicola, Jarke 99] Matthias Nicola, Matthias Jarke: “*Increasing the Expressiveness of Analytical Performance Models for Replicated Databases*”, International Conference on Database Theory, ICDT'99, Jerusalem, January 1999.

[Oracle 93] „*Oracle 7 Symmetric Replication*“, White paper, Oracle Corporation, September 1993.

[Pacitti, Simon 98] E. Pacitti, Eric Simon: „*Update Propagation Strategies to Improve Freshness of Data in Lazy Master Schemes*“, Technical Report, INRIA Rocquencourt, France, 1997/98.

[Poledna 94] Stefan Poledna: „*Replica Determinism in Distributed Real-Time Systems: A Brief Survey*“, Journal on Real-Time Systems, Vol. 6, 1994, pp. 289-326.

[Rabinovich et al. 96] M. Rabinovich, N. Gehani, A. Kononov: „*Scalable update propagation in epidemic replicated databases*“, Proceedings of the 5th International Conference on Extending Database Technology, pp. 207-222, March 1996.

[Shasha 92] D. Shasha: “*Database Tuning: A Principled Approach*”, Prentice Hall, 1992.

[Shyy et al. 95] Yuh-Ming Shyy, H. Stephen Au-Yeung, C.P. Chou. „*VERSANT Replication*“, ACM-SIGMOD International Conference on Management of Data, May 1995.

[Sidell et al. 96] Jeff Sidell, P.M. Aoki, A. Sah, C. Staelin, M. Stonebraker, A. Yu: „*Data replication in Mariposa*“, Proceedings 12th International Conference on Data Engineering, pp. 485-494, 1996.

[Skeen, Stonebraker 83] Dale Skeen, Michael Stonebraker. „*A Formal Model of Crash Recovery in a Distributed Systems*“, IEEE Transactions on Software Engineering, 9, pp. 219-228, May 1983.

[Stonebraker 79] Michael Stonebraker. „*Concurrency Control in Distributed Ingres*“, IEEE Transactions on Software Engineering, 5(3):188-194, 1979.

[Sybase 94] „*SYBASE Replication Server*“, Technical Overview, Sybase Corporation1994.

[Tandem 96] „*RDF and RDF/MP*“, Tandem NonStop Servers Product Description, Tandem 1996.

[Tikkanen 93] M. Tikkanen: „*Objects in a telecommunications oriented database*“, Proceedings of the Conceptual Modelling and Object-Oriented Programming Symposium, 1993.

Appendix: Formal Description of ADR

This Appendix gives a formal description of ADR and proves that asynchronous propagation is sufficient with respect to consistency. It is shown that a step in the schedule at a secondary copy observes a state of the data which might have been seen by the step if executed in the schedule at the primary copy.

We use the serializability theory for centralized database systems. The problem is, that in a general distributed database, transactions are submitted in parallel at multiple sites and there is no absolute time and therefore no global ordering of operations. However, the ADR mechanism is a special case which avoids these ordering problems such that the "centralized" theory is valid: Firstly, ADR employs the primary copy approach. This implies that reproduction transactions which update a certain secondary copy are not generated at multiple sites but only at the respective primary site. Hence, ordering of reproduction transactions for individual secondary copies is ensured by timestamps generated by the clock at the primary site. Secondly, if a write step updates primary copies at multiple sites, ADR employs 2PC which synchronizes the participating sites. Thirdly, if a read requires full (partition external) consistency, it is executed in a write step at the primary copy.

Definition 1 (partitioned database)
A partitioned database (OBJ,P) *consists of a set of objects OBJ and a decomposition P into mutually disjoint subsets with* $P = \{u,v,w,...\}$ *where* $OBJ = \bigcup_{u \in P} u$.

The abstraction from transactions as executable programs to a sequence of database actions is commonly known as the *read-write model*. This can be formalized by the following definition:

Definition 2 (transaction)
Let $r^i(x_v)$ *denote read data element* x_v, $w^i(x_v)$ *denote write data element* x_v, a^i *denote abort transaction and* c^i *denote commit transaction. Then a transaction* $T^i = (O^i, <<^i)$ *consists of*

- *a finite set of operations* $O^i = \{ o^i_1(x_p),..., o^i_n(x_q)\} \cup \{s^i\}$ *called actions,*
  where $o^i_j(x_v) \in \{r^i(x_v), w^i(x_v)\}$, $s^i \in \{c^i, a^i\}$, $n < \infty$, $1 \le j \le n$
- *a partial ordering* $<<^i \subseteq O^i \times O^i$ *with*
  - $\forall\ o^i_j(x_v) \in O^i :\ o^i_j(x_v) <<^i s^i$
  - $\forall\ o^i_k, o^i \in O^i, k \neq l :\ o^i_k \le o^i \vee o^i_k \ge o^i$

Read and write operations appear interleaved in different concurrent transactions. The ordering of these operations is called a schedule.

Definition 3 (schedule)
A schedule $s = (\tau, <)$ *consists of*

- *a set* $\tau = \{T^1, ..., T^n\}$ *of transactions.* $O(s) := O^1 \cup ... \cup O^n$
- *a partial ordering* $< \subseteq O(s) \times O(s)$ *with:*
  $\forall\ o^i_k, o^j_l \in O(s), (i,k) \neq (j,l) :\ o \le o^j_l \vee o \ge o^j_l$

Operations on the same data element are recognized to be in conflict, if one of them is of type write. This property is used to define the dependency graph.

Definition 4 (dependency graph)

Let $s = (\tau, <)$ be a schedule. Then the dependency graph $G(s)$ is a directed graph having

- *transactions from τ as nodes*
- *an edge from T^i to T^j, if T^i and T^j have a conflicting operation*

In our replicated database, transactions may work on any partitions reading or writing data. For our execution model we define the following:

Definition 5 (read and write space)

The working space $P(T^i)$ of a transaction $T^i = (O^i, <<^i)$ is the set of partitions accessed:
$P(T^i) = \{u \in P \mid \exists\ o\ (x_v) \in O^i \wedge x_v \in u\}$.

The write space $WP(T^i)$ of a transaction $T^i = (O^i, <<^i)$ is the set of partitions where partition external consistency is needed: $\{u \in P \mid \exists\ w_n(x_v) \in O^i \wedge x_v \in u\} = WP(T^i) \subseteq P(T^i)$

The read space $RP(T^i)$ of a transaction $T^i = (O^i, <<^i)$ is the set of partitions where partition internal consistency is sufficient. $RP(T^i) = P(T^i) \setminus WP(T^i)$.

Definition 6 (step)
A transaction $T^i = (O^i, <<^i)$ in a partitioned database (OBJ, P) is split into at most one write step $T^{i,W} = (O^{i,W}, <<^{i,W})$ with $W = WP(T^i)$, and zero or more read steps $T^{i,u}$ with $u \in RP(T^i)$.

$$
\begin{aligned}
O^{i,W} &= \{ o_n(x_v) \in O^i \mid \exists w \in WP(T^i) : x_v \in w\} \cup \{c^{i,W}\}, && \text{if } c^i \in O^i \\
O^{i,W} &= \{ o_n(x_v) \in O^i \mid \exists w \in WP(T^i) : x_v \in w\} \cup \{a^{i,W}\}, && \text{if } a^i \in O^i \\
O^{i,u} &= \{ o_n(x_v) \in O^i \mid x_v \in u\} \cup \{c^{i,u}\}, && \text{if } c^i \in O^i \\
O^{i,u} &= \{ o_n(x_v) \in O^i \mid x_v \in u\} \cup \{a^{i,u}\}, && \text{if } a^i \in O^i
\end{aligned}
$$

$<<^{i,W}$ is the restriction of $<<^i$ on $O^{i,W}$ and $<<^{i,u}$ is the restriction of $<<^i$ on $O^{i,u}$ taking into account the new operations $c^{i,u}$, $c^{i,W}$, $a^{i,u}$ and $a^{i,W}$.

Write steps directly affect only primary copies and are executed by the database system with classical concurrency control ensuring serializability. Secondary copies are updated by a so-called reproduction transaction which is constructed based on the write steps using the reproduction function.

Definition 7 (reproduction function)
Let θ denote the set of all possible transactions. A reproduction function $rf: \theta \times P \rightarrow \theta$ generates for every committed write step T^i and every partition u a reproduction transaction T^i_u, working only on partition u, i.e. $RS(T^i_u) \subseteq u$ and $WS(T^i_u) \subseteq u$. The reproduction function is extended on sets of write steps τ as $rf(\tau,u) := \{rf(T^i,u) \mid T^i \in \tau\}$.

Definition 8 (formal reproduction)
Let $T=(O, <<)$ be a committed write step and $u \in P$ be any partition. Then we define the formal reproduction as $rf(T, u) := (O', <<|_o)$ where $O' = \{w(x) \mid x \in u\} \cup \{c\}$.

So the formal reproduction constructs a reproduction transaction containing all write operations of the write step accessing the partition *u* in the same order as they appear in the write step. Thus, it repeats exactly the data modifications of the primary copy on the secondary copies. For the following it is sufficient to ensure the correctness of the reproduction function, yielding another possibility for optimization.

Definition 9 (correct reproduction function)
A reproduction function rf is called correct, if for

- *any set of committed write steps* $\tau = \{T^1, T^2, \ldots\}$
- *any partition* $u \in P$
- *any single read step* T_u*, reading only data of partition u*
  (*i.e.* $RS(T_u) = u$, $WS(T_u) = \varnothing$)
- *any serial schedule* $s = (\tau \cup \{T_u\}, <)$*, made of the write steps and the one read step, and* $s_u = (rf(\tau, u) \cup \{T_u\}, <_u)$*, made of the additional reproduction transactions, the write steps and the one read step having the same order of transactions:*
  $\forall T^1, T^2 \in \tau : T^1 < T^2 \Leftrightarrow rf(T^1, u) <_u rf(T^2, u)$
  $\forall T^1 \in \tau : T^1 < T_u \Leftrightarrow rf(T^1, u) <_u T_u$
  $\forall T^1 \in \tau : T_u < T^1 \Leftrightarrow T_u <_u rf(T^1, u)$

it is valid that T_u in s reads the same value as in s_u.

Definition 10 (correct execution of reproduction transactions and read steps)
Given $s = (\tau, <)$ *as a conflict serializable schedule of write steps,* $u \in P$ *as any partition,* T_u *a read step accessing only data of the partition u* ($RS(T_u) \subseteq u$ *and* $WS(T_u) = \varnothing$)*, and a reproduction function rf. A conflict serializable schedule* $s_u = (rf(\tau,u) \cup \{T_u\}, <_u)$ *is called* ***correct execution of reproduction transactions and the read step*** *if:*

$$\forall\, T^1, T^2 \in \tau : T^1 \rightarrow T^2 \text{ in } G(s) \;\Rightarrow\; rf(T^1, u) \rightarrow rf(T^2, u) \text{ in } G(s_u)$$

Theorem 1 (a read step in the reproduction schedule reads the same values as in the write step schedule)
Given

- *rf is a correct reproduction function*
- *each transaction has at most one write step*
- $\tau = \{T^1, T^2, \ldots\}$ *a set of committed write steps*
- $s = (\tau, <)$ *a conflict serializable schedule of* τ
- $u \in P$ *any partition*
- T_u *a read step accessing only data of u ,i.e.* $RS(T_u) \subseteq u$ *and* $WS(T_u) = \varnothing$
- $s_u = (rf(\tau,u) \cup \{T_u\}, <_u)$ *as conflict serializable, and a correct execution of the reproduction transactions and the read step*

Then there exists a serial schedule $s' = (\tau \cup \{T_u\}, <')$*, in which* T_u *reads the same values as in* s_u *and where the sub-schedule* $s'|_\tau = (\tau, <'|_\tau)$ *has the same dependency graph as s. This means the read step executed in the reproduction schedule* s_u *reads the same values as in the write step schedule.*

PROOF
1. There exists a serial schedule $s_u' = (rf(\tau,u) \cup \{T_u\}, <'_u)$ with $G(s_u) = G(s_u')$ because s_u is conflict serializable. T_u in s_u reads the same values as in s_u'.

2. T_u in s_u' reads the same values as in the serial schedule $s' = (\tau \cup \{T_u\}, <')$ with the following ordering as *rf* is a correct reproduction function:

$$\forall\, T^1, T^2 \in \tau : rf(T^1, u) <_u' rf(T^2, u) \;\Leftrightarrow\; T^1 <' T^2$$

$$\forall T^1 \in \tau : rf(T^1, u) <_u' T_u \Leftrightarrow T^1 <' T_u$$
$$\forall T^1 \in \tau : T_u <_u' rf(T^1, u) \Leftrightarrow T_u <' T^1$$

3. s' and s restricted to τ have the same dependency graph.

(a) $\forall T^1, T^2 \in \tau : T^1 \to T^2 \in G(s) \Rightarrow T^1 \to T^2 \in G(s')$
The following must be valid:
$\forall T^1, T^2 \in \tau : T^1 \to T^2 \in G(s) \Rightarrow T^1 \to T^2 \in G(s') \vee T^2 \to T^1 \in G(s')$
because T^1 and T^2 contain conflicting operations through $T^1 \to T^2 \in G(s)$.

Assumption:
$\exists T^1, T^2 \in \tau : T^1 \to T^2 \in G(s) \wedge T^2 \to T^1 \in G(s')$
$\Rightarrow \exists T^1, T^2 \in \tau : rf(T^1, u) \to rf(T^2, u) \in G(s_u) \wedge T^2 <' T^1$
$\Rightarrow \exists T^1, T^2 \in \tau : rf(T^1, u) <_u' rf(T^2, u) \wedge rf(T^2, u) <_u' rf(T^1, u)$
Contradiction!

(b) $\forall T^1, T^2 \in \tau : T^1 \to T^2 \in G(s') \Rightarrow T^1 \to T^2 \in G(s)$
The following must be valid:
$\forall T^1, T^2 \in \tau : T^1 \to T^2 \in G(s') \Rightarrow T^1 \to T^2 \in G(s) \vee T^2 \to T^1 \in G(s)$
because T^1 and T^2 contain conflicting operations through $T^1 \to T^2 \in G(s')$.

Assumption:
$\exists T^1, T^2 \in \tau : T^1 \to T^2 \in G(s') \wedge T^2 \to T^1 \in G(s)$
$\Rightarrow \exists T^1, T^2 \in \tau : rf(T^2, u) \to rf(T^1, u) \in G(s_u) \wedge T^1 <' T^2$
$\Rightarrow \exists T^1, T^2 \in \tau : rf(T^2, u) \to rf(T^1, u) \wedge T^1 <_u' T^2$
Contradiction!

❑

Corollary 1 (Conflict-Serialization of the read steps)
All read steps read the same values as in a conflict-serializable write step schedule if

1. *the reproduction function is correct*
2. *every write step schedule s is conflict serializable*
3. *every schedule of reproduction transactions and read steps is correctly conflict-serializable executed*

PROOF
3. implies that every schedule of reproduction transactions and a read step is conflict serializable and correctly executed. The proposition follows by Theorem 1, 1. and 2. Informally, this lemma means that a transaction always looks at a state of the database which it might have seen if executed at the primary copy.
❑