17

A Highly Available, Real-Time Database System for Telecom Applications

S.-O. Hvasshovd, Ø. Torbjørnsen, S. E. Bratsberg
Telenor Research and Development
N-7005 Trondheim, Norway
email: {Svein-Olaf.Hvasshovd,Oystein.Torbjornsen,Svein.Bratsberg}@fou.telenor.no
P. Holager
SINTEF Communication and Informatics
N-7034 Trondheim, Norway
email: Per.Holager@delab.sintef.no

Abstract
Due to new data-intensive telecommunication services and mobility networks, databases have been introduced as parts of telecommunication networks. Compared with traditional use of databases, telecom databases must fulfill very tough requirements on availability, response time, and throughput. ClustRa is a telecom database developed to run on standard workstations interconnected by standard high performance networks. To meet the throughput and real-time response requirements, ClustRa is a main-memory database with main-memory logging. Transactions are executed in parallel. To meet the availability requirements, we use a 2-safe replication scheme over two sites with independent failure modes, a novel declustering strategy, early detection of failures with fast takeover, and by on-line self-repair and maintenance. This paper gives an overview of ClustRa and includes a set of performance measurements.

Keywords
Telecommunication, main-memory, real-time, distributed, parallel, high availability

1 INTRODUCTION

Traditional telephony services are handled by monolithic, digital switches. They have over time been stuffed with increasing amounts of functionality, e.g., for management of routing, terminals, subscribers and charging. A digital switch typically contains millions of lines of software. It is not allowed to be out of service for more than two to three minutes per year, corresponding to availability class five (Gray, 1992). The large amount of software combined with the required high availability results in long lead time for introduction of new services and a dominance of switch suppliers over service providers.

In the 80's intelligent networks (IN) were introduced to support new types of telecom services, e.g., terminal mobility (UPT), virtual private nets, and credit card calling. Most IN services are required to have the same service availability as traditional telephony services. IN are designed for rapid development and deployment of new services and to give service operators control over service development. To obtain this goal, services are built by service programs using basic functions supported by specialized servers (SCPs) in the network. The effect is that functionality that was buried invisibly inside digital switches becomes open, modularized, and allocated as servers on multiple platforms. One entity that appears in this architecture is the telecom database.

Telecom databases are classically used in various types of call routing, where transactions read one record, demand 5 to 50 milliseconds response time, availability class five, and 10 to 10.000 TPS. Examples are mapping from phone number to terminal in a digital switch, and from a universal phone number to a physical phone number in UPT. Update transactions are less than 10% of the transaction volume, and demand from 50 milliseconds to a few seconds response time, with availability class five. Mobile telephony implies update transactions with 10 to 20 millisecond response time because a user should experience no longer glitches than 100 milliseconds when a mobile terminal is handed over from one switch (MSC) to another. Durable connections which imply crash atomic call status, demand update transactions with similar response time as mobile telephony.

To meet the total combination of requirements from IN and mobility networks we have designed and developed the ClustRa telecom database. As a consequence, all performance requirements refer to transactions updating four tuples. The following are the main project goals: 1) Response time for four-tuple update or lighter transactions of maximum 15 milliseconds for at least 95% of the transactions; 2) Scalable throughput with an upper limit of at least 1000 TPS; 3) Class five availability.

These requirements are approached by a combination of old and well known techniques, together with new techniques developed primarily to meet the availability goal. To meet the response time, ClustRa employs a main-memory database for real-time data, main-memory logging, and parallel intra-transaction execution. A parallel database design is used to achieve scalable transaction throughput. To meet class five availability we use a 2-safe replication synchronization over two sites, and on-line self-repair. ClustRa uses the relational model, but can also support object oriented models. The system is also designed to support traditional on-line and decision support transaction types through the use of traditional database buffering and a flexible record structure.

The paper is organized as follows: Related work is briefly surveyed in Section 2. Section 3 presents the ClustRa architecture and how it executes distributed transactions. The ClustRa availability management is presented in Section 4. The detection and masking of node failures are emphasized. Section 5 gives an overview of the ClustRa logging and recovery method. Measurements of transaction response and throughput together with take-over time are presented in Section 6. Section 7 concludes the paper.

2 RELATED WORK

Switch manufacturers have been the main developers of telecom databases. These systems have barely been documented in the research literature. They were tailored to the needs of routing applications in digital switches. As a consequence, they support very fast reading of a few records, and most provide dirty read. Some systems are pure main-memory databases, others have background disk support. Update transactions involve write-ahead logging to disk, and the response time is therefore longer than a disk access. The throughput rate for update transactions is rather low. Some systems are centralized, some are parallel. The parallel systems seem to give scalable throughput growth. They differ on availability attention and implementation. Some rely entirely on fault-tolerant hardware, others have implemented this in software using multiple loosely synchronized main-memory replicas of tables. A hot stand-by replica becomes primary in case the primary fails. Automatic repair is realised by producing a new replica for one that has been lost. On-line software upgrade is not supported. The relational model is used by most systems.

During the last years new telecom databases have appeared (Ahn, 1994). One of these is the Dalí system from AT&T Bell Labs (Jagadish, 1994). This is a single node site main-memory system with background disk support that is tailored to routing applications. It supports more flexible record structures than most older products. It uses a 1-safe hot spare system with a system as the atomic failure unit. By utilizing hot spares, Dalí indicates ability to support schema and software changes without bringing the entire system down. Dalí does not have throughput scalability given its current centralized architecture, nor does it support on-line automatic repair, which is imperative to achieve class five availability.

Hewlett-Packard Laboratories has been developing Smallbase (Heytens, 1994). The system is designed for the throughput and response time characteristics of IN transactions. A main goal is to achieve transaction scale-up using commodity hardware and Unix. Like Dalí, Smallbase uses a 1-safe hot spare system.

Nokia has developed TDMS for use in switching and mobile systems (Tikkanen, 1992, Tikkanen, 1993). The system runs on dedicated hardware and basic software. The focus is on response time and throughput of read transactions. Throughput scale-up and high availability have not been catered for beyond the use of fault-tolerant hardware.

Some commercial SQL databases are used within IN and some mobile applications. Some SQL systems meet throughput and response time requirements for read-only transactions, but their applicability to mobile and switch applications are limited by their longer response time for update transactions caused by disk logging. Availability has been achieved through system pairs and manual repair. Limited attention has been given to aspects like on-line schema upgrades.

3 ARCHITECTURE

3.1 Hardware and Software Platform

The ClustRa database management system uses a shared-nothing hardware model to support fault-tolerance. Each *node* of the database system is a standard Unix workstation, with inter-node communication based on ATM. The purpose of the project is partly to show that it is possible to meet the requirements of a telecom database using standard, off the shelf hardware and operating system.

Each node is an atomic failure unit. Nodes are grouped into *sites*, which are collections of nodes with correlated probability of failure. Sites are failure independent with respect to environment and operational maintenance. Logically, the database system consists of a collection of interconnected nodes that are functionally identical and act as peers, without any node being singled out for a particular task. This improves the robustness of the system and facilitates the maintenance and replacement of nodes. Figure 1 shows an architecture with two sites, each having four nodes. Each site has a replica of the database. Each node at a site is connected to an ATM switch, which again is connected to the switch at the other site. The switches have external connections.

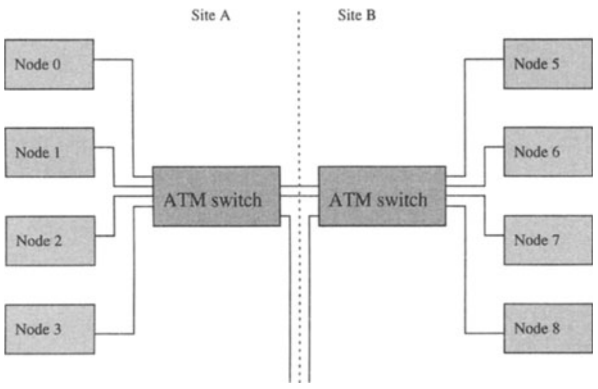

Figure 1 An architecture with two sites.

3.2 Storage Manager

Unlike other telecom databases, ClustRa is a traditional database server. This means that it manages a buffer of data with a disk-based layout in blocks; it has a B-Tree access method, a semantically rich two-phase record locking protocol, and it has a two-level logging approach. However, it is main-memory-based in the sense that tables may be declared to reside in main memory. Unlike pure main-memory databases (Heytens, 1994, Jagadish, 1994), this allows for many classes of queries and transactions, not limited to those requiring real-time response. This is important in a telecom database, due to multiple services having diverse characteristics with respect to response time and data volume.

Scalability and high throughput is achieved by a distributed architecture. A table may be distributed onto different nodes by horizontal fragmentation, according to either a hash or a range partitioning algorithm. High availability is achieved by replication over several sites. We are using an asymmetric replication scheme, where there always will be one primary replica of a (horizontal) fragment of a table, but there may be several hot stand-by replicas. The node of the primary replica will always be the one executing the request for a specific record found in that replica. The hot stand-by replicas will be kept consistent by redoing log records that are shipped from the primary node. Each node in the system may be primary for some fragments, and hot stand-by for other fragments. This facilitates load balancing both during normal processing and during node failure where *takeover* must take place, i.e. the hot stand-bys will become primary. The number of hot stand-by replicas are dependent on the desired availability level. For our requirements it is enough with one hot stand-by replica (Torbjørnsen, 1995).

The data model of ClustRa provides a basis for a relational system. It supports variable length records identified by primary keys and organized in tables. Records may be accessed by primary keys or sequentially, and they are stored in fixed sized blocks according to the block size in the underlying secondary storage. Tables are stored in files, which currently are organized as B-Trees. This holds also for resident, internal administration data, like the free block management, the resident part of the distributed log, and the file directory, which is a mapping from file identifiers to root block identifiers. By using B-Trees also for internal data structures, we have reduced the code volume of the system. We have chosen to use a B-Tree access method due to its generality – it gives sufficient performance both with respect to sequential and direct record access. The buffer manager holds copies of blocks residing on disk with three different priorities:

- *Real-time*: the block always resides in main memory.
- *Random*: the block will be accessed randomly and is kept in the buffer according to a LRU policy.
- *Sequential*: the block will be accessed sequentially. This is a hint to the buffer manager to prefetch the next blocks in the same file and to make used blocks available for replacement.

3.3 Execution of Distributed Transactions

Each node has a number of services: A transaction controller, a database kernel, a node supervisor, and an update channel. The transaction controller receives requests from clients to execute certain precompiled procedures, or it receives user code which it compiles into the internal code format. It coordinates transactions through a two-phase commit protocol. The kernel has the main database storage manager capabilities, like locking, logging, access methods, block and buffer management. It receives code to be interpreted from a transaction controller. The update channel is responsible for reading the log and for shipping log records to hot stand-by nodes. The node supervisor is responsible for collecting information about the availability of different services, and for informing about changes.

In Figure 2 we have illustrated the execution of a simple transaction by showing the processes involved. There are two sites with two nodes each. The transaction controller (T0) receives the client request, and thus become the primary controller. It has a hot stand-by controller (T5) on another node, which is ready to take over as primary if T0 fails. T0 uses the distribution dictionary to find the nodes holding the primary and the hot stand-by replicas of the records in question. For this particular transaction, K1 has the role of a primary kernel and K6 the role of a hot stand-by kernel. T0 sends the operations to be executed *piggybacked* on the start transaction command to K1. Simultaneously, it sends K6 a start transaction command together with the number of log records to receive from K1. The update channel (U1) reads the log of K1 and ships the log records for this transactions to K6, where they are stored in the log and redone. When T0 has received ready (and possible return values) from both kernels and an ack from the hot stand-by controller, it gives an *early answer* to the client. The second phase of the commit includes sending commit messages to the two kernels involved. When both kernels have responded to T0 with done, the transaction is removed from T5 and T0.

The picture is simplified, because when a transaction accesses several records, there may be several primary and hot stand-by kernels executing the transaction in *parallel*.

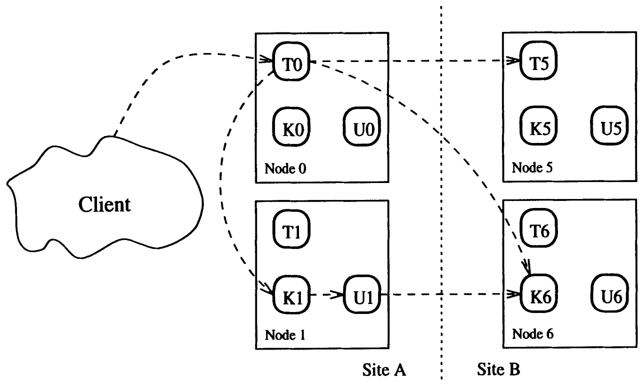

Figure 2 The services involved in a simple transaction.

4 HIGH AVAILABILITY AND SELF-REPAIR

To support high availability we rely on data replication and allocation of primary and hot stand-by fragment replicas to nodes with independent failure modes. Additionally, ClustRa balances the load on the nodes both during normal operation and failures, and it supports *self-repair*.

ClustRa applies an I-am-alive protocol internally inside a node and between nodes to discover failed services or nodes. All processes on a node are polled repeatedly by a node supervisor to detect their state. The I-am-alive protocol between the nodes is organized as a circle. Each node sends and receives I-am-alive messages from both its neighbors in the circle. In addition to discovery of failed nodes and services, the I-am-alive protocol is used whenever a node is changing the set of provided services. It is also used to determine which unavailable nodes each available node is responsible for attempting to restart at regular intervals.

The I-am-alive system is fully distributed with no centralized knowledge. Hence, a *virtual node set protocol* is used to maintain a consistent set of available nodes (El Abbadi, 1988). If consecutive I-am-alive messages are missing from one of the neighbors in the circle, the protocol is activated. The node supervisor sends a build node set message to all known nodes. Other nodes should respond with their services. If a node has not responded within a certain number of resends, the node is assumed to be down, and a new node set is distributed.

After a node failure it is important to perform a fast take-over to achieve high availability. Given the low probability of lost messages and the short message queues in the ATM technology, we minimize the number of missing I-am-alive messages before the virtual node set protocol is activated. This shortens the interval from a failure happens to it is reacted upon. To minimize the delay caused by updating the distribution dictionary, it is cached at each node and the current node set is used as a filter to it. The distribution dictionary itself is modified outside critical path. To minimize unavailability of replicas changing their role to primary, locks are maintained (but not effective) to the hot stand-by replicas. These replicas become available immediately after redoing the hot stand-by log records arriving before the new node set. In-flight transactions being active at a crashed node are rolled back, while those losing their hot stand-by controller continue. To minimize the window where the system is vulnerable to double faults, the log where only one replica exists, is copied to a node at the other site. Together with the main-memory logging to two sites with independent failure modes, this is as reliable as traditional logging to disk (Torbjørnsen, 1995).

When a takeover has happened, a recovery of the failed node is started. If it does not recover within a certain period of time, ClustRa starts its *on-line, non-blocking self-repair.* New replicas of its fragments, possibly subfragmented, are produced fuzzily, and they are then caught up with the use of distributed logs (Hvasshovd, 1991).

A goal of ClustRa is to achieve high availability and load balancing, both during normal processing and upon failures. To fulfill this goal, a new strategy for placement of replicated fragments, *minimum intersecting sets declustering*, has been developed (Torbjørnsen, 1995). The principle of this methodology is that the largest cardinality of any intersection of the sets of fragments allocated to different nodes, should be minimized. It achieves low unavailability by fast takeover and self-repair. A takeover is made fast by tuning the system to not involve more nodes than necessary. Self-repair is made fast by exploiting spare capacity of non-failed nodes.

5 TWO-LEVEL LOGGING AND RECOVERY

ClustRa combines two logging methods: a distribution transparent method for record operations and a physiological method for node-internal operations.

5.1 The Distributed Log

The *neighbor write-ahead logging* developed by Hvasshovd (1992) is applied to record operations. This is a main-memory logging technique where transaction commit does not force log to disk. A log record must be written to two nodes with independent failure modes, before the effect of an update operation is allowed to be reflected on disk, and before the transaction commits. The log shipping serves in addition as the loose replication synchronization scheme (Cristian, 1990).

This technique saves the delay inherent in forcing log to disk at commit time. We do this by taking advantage of modern communication technology, where the delay in writing to main memory on a neighbor node is much lower than the delay in writing to disk. Since we exploit the log shipping for replication as well, this comes almost for free. The distributed logging uses a compensation log record (CLR) policy. CLRs are produced by the primaries and contain complete redo information and reference to the log record it compensates. Only a single CLR can be produced per non-CLR (Mohan, 1992).

By having *logical*, primary-key based redo and undo we allow for subfragmentation of a hot stand-by replica as compared with its corresponding primary. This allows for encapsulation of block size and access method in each node. Of the same reason state-identifiers reflecting the sequence of operations executed to a record are connected to records instead of as traditionally to blocks. State-identifiers are generated at the primaries with unique values within a fragment. They are included in the log records and are reflected to the hot stand-bys through redo processing. Therefore, two replicas of a record with the same state-identifier reflect the same state independently of the fragment, replica, and node the record is located at.

5.2 The Node-Internal Log

Operations for access methods, free block management, and file directory are node-internal and they are implemented transactionally. These operations are logged in a node-internal log. A physiological, single CLR policy with just reference to the compensated non-CLR is used (Gray, 1992). The node-internal log is disk-based and is not shipped to any other node. To be able to undo a node-internal operation a node-internal log record must be produced in main memory before the corresponding operation is executed. The log record must have been flushed to disk before the effect of the operation is reflected on disk. A sequence may be imposed on writing blocks to disk to avoid logging records moved in a block split. Committing a node-internal transaction does not involve any block flush, because redoing the distributed log to the node will establish an equivalent node-internal state. This synchronization protocol between the distributed and the node-internal logging avoids altogether introducing log-related disk flushes in the time-critical transaction execution.

5.3 Node Crash Recovery

The node crash recovery method of ClustRa distinguishes among different degrees of corruption at a node. When the disk is corrupted, the complete database is *reloaded* from other nodes. When the contents of main memory is garbled, a recovery based on log records and database from disk is done. When only parts of main memory are corrupted, a recovery based on main memory is done. The decision on using disk or main-memory recovery is based on checksums on the internal administration data. If the checksums on the buffer access structure and the log access structure are found to be in order, a main-memory recovery may be done. A node recovery based on main memory involves just undoing the eventual ongoing node-internal transactions and undoing the record operations that were not reflected on any other node before the crash occurred. A disk-based recovery performs a redo followed by an undo recovery from the stable node-internal log, before a redo recovery is executed based on the distributed log shipped from nodes with primary fragments for those stored at the recovering node. If during a main-memory-based recovery a block is found to be corrupted upon access, i.e. records or administration data inside a block is broken, a partial recovery is performed by reading the block from disk, redoing the node-internal log regarding this block, and then the distributed log.

6 MEASUREMENTS

All measurements presented here were taken on eight HP9000/735 125 running HPUX 9.0.5 and four two-processor HP9000/I70 running HPUX 10.0.0. The nodes were interconnected with a FDDI network. The multiprocessor computers were used such that each of them roughly corresponds to two singleprocessor nodes with all communication going through the FDDI network. The system was developed on Sun Sparcstations interconnected by an ATM switch, but for the sake of benchmarking it was ported to the configuration described above. The porting was done by one person in less than one day.

6.1 Response Time and Throughput

To measure the scale-up of the system we created a transaction updating four records in four different tables. The tables were fragmented to all nodes, both as primary and as hot stand-by. To scale the load, each node has a separate client running this four-tuple update transaction back-to-back, i.e., once it receives the result of the previous transaction, it ships the next. Thus, when we add a node, we also add a client.

Figure 3 shows the minimum, average, and maximum response times when scaling up the system from two to 16 nodes. The statistics is gathered over a period of 120 seconds. All response times are measured in milliseconds. In these measurements the percentage of transactions responding within 15 milliseconds decreases from 100% when having 2 nodes, to 89.7% when having 16 nodes. However, the average response time seems to reach an asymptote.

Figure 4 shows the measurements for throughput when scaling the system from two to 16 nodes. From these figures we can see that this configuration has a maximum throughput of 1052 2-safe four tuple update TPS. In both figures we can see a slight zig-zag pattern. This is an effect of the dual processor HP9000/I70 competing for memory resources and FDDI interconnect.

6.2 Takeover

Fast takeover is necessary to ensure high availability. In our measurements we used a two node configuration. One node holds the primary kernel and the hot stand-by controller.

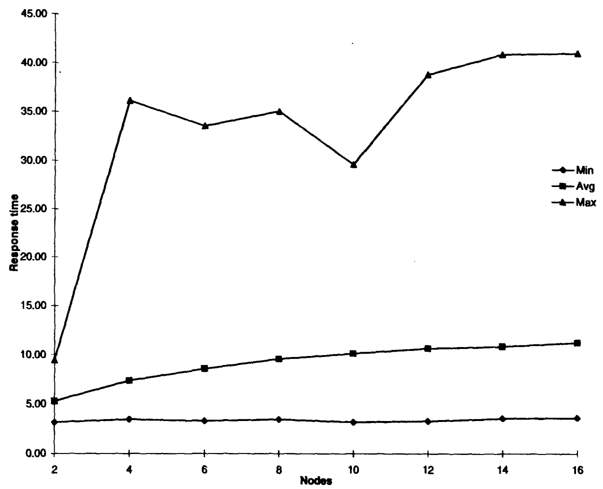

Figure 3 Minimum, average, and maximum response times (milliseconds) when scaling up the system for four-tuple update transactions.

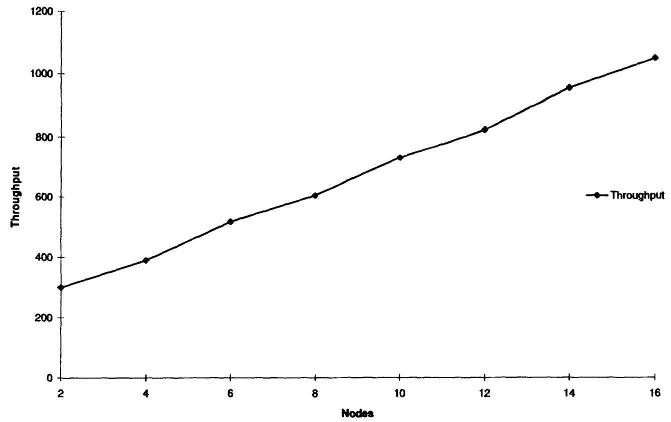

Figure 4 Throughput when scaling up the system for four-tuple update transactions.

The other holds the hot stand-by kernel and the primary controller. The former node was stopped by sending it a UNIX signal. The client sends transactions back-to-back to the controller and receives either committed or aborted status. The time is taken between the receipt of two committed transactions, where there was a node crash in between.

The measured takeover times was (average, minimum, maximum): (330, 265, 379) milliseconds. In these measurements the I-am-alive messages are sent between nodes every 50 millisecond. If a neighbor node in the I-am-alive circle has not sent any I-am-alive messages within 100 milliseconds, the virtual node set protocol is started. The node supervisor discovering the lacking I-am-alive message builds a new node set by asking all known nodes about their services. The nodes which have not responded within 50 milliseconds, is questioned once more. If they have not answered within another period of 50 milliseconds, they are assumed to be down, and a new node set is distributed. In average, the two phases take 225 milliseconds (125 + 100). The rest of the time is used in the takeover itself.

7 CONCLUSIONS AND FURTHER WORK

We know there there are several other approaches addressing the response time and throughput requirements, but we do not know of any other approach which address all three requirements of response time, throughput, *and* high availability. Until now the ClustRa project has approached two of its main goals for response time and throughput, and it has partially met the third goal for high availability. The *response time* requirement is very close to be met on this configuration of the system. The main technique used to meet this goal is the main-memory database *with* main-memory logging. main-memory logging is possible by writing the log to the main memory of another node with an independent failure mode. The *throughput* seems to scale linearly, and the goal of 1000 TPS has been reached. The main components for *high availability*, fast take-over, takeback and on-line self repair, have been implemented.

Further work includes estimation of long term availability by accelerated testing. This includes failure rate prediction and fault injection to estimate the error handling coverage. In the next stages we will put effort in further enhancements of availability. We are developing a protocol for on-line non-blocking upgrade of dictionary data, a set of algebra methods for incremental on-line modification of data as a result of schema updates, and support for upgrade of basic software and disk structures.

REFERENCES

Ahn, I. (1994) Database issues in telecommunications network management. *Proceedings of ACM/SIGMOD*, 37-43.

Cristian, F. (1990) Understanding fault-tolerant distributed systems. *Research report, IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120-6099, USA.*

El Abbadi, A., Skeen, D. and Cristian, F. (1988) An efficient, fault-tolerant protocol for replicated data management. *M. Stonebraker (editor), Readings in Database Systems, Morgan Kaufmann Publishers*, 259–273.

Gray, J. and Reuter, A. (1992) Transaction Processing: Concepts and Techniques. *Morgan Kaufmann Publishers.*

Heytens, M., Listgarten, S., Neimat, M. and Wilkinson, K. (1994) Smallbase: A main-memory DBMS for high-performance applications (release 3.1). *Hewlett-Packard Laboratories, Palo Alto, CA, USA.*

Hvasshovd, S. O., Sæter, T., Torbjørnsen, Ø., Moe, P. and Risnes, O. (1991) A continuously available and highly scalable transaction server: Design experience from the HypRa project. *Proceedings of the 4th International Workshop on High Performance Transaction Systems.*

Hvasshovd, S. O. (1992) HypRa/TR: A tuple oriented recovery method for a continuously available distributed DBMS on a shared nothing multi-computer. *PhD thesis, The Norwegian Institute of Technology.*

Jagadish, H. V., Lieuwen, D., Rastogi, R. and Silberschatz, A. (1994) Dalí: A high performance main memory storage manager. *Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile (VLDB '94)*, 48–59.

Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H. and Schwarz, P. (1992) ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead locking. *ACM Transactions on Database Systems*, **17(1)**, 94–162.

Tikkanen, M. (1992) TDMS: An embedded real-time main-memory database management system. *Proceedings of Embedded and Real-Time Systems, The Finnish Artificial Intelligence Society.*

Tikkanen, M. (1993) Objects in a telecommunications oriented database. *Proceedings of the Conceptual Modelling and Object-Oriented Programming Symposium.*

Torbjørnsen, Ø. (1995) Multi-site declustering strategies for very high database service availability. *PhD thesis, The Norwegian Institute of Technology.*

BIOGRAPHY

Svein-Olaf Hvasshovd is a senior research scientist at Telenor Research and Development. He has been doing research within database systems and recovery in parallel database system for more than 15 years. He received a Siv.ing. (MSc) in computer science in 1980 and a Dr.ing. (PhD) in computer science in 1992 from the Norwegian Institute of Technology.

Øystein Torbjørnsen is a research scientist at Telenor Research and Development. His research has been focused on parallel database computers and highly available database systems. He received a Siv.ing. (MSc) in computer science in 1987 and a Dr.ing. (PhD) in computer science in 1995 from the Norwegian Institute of Technology.

Svein Erik Bratsberg is a research scientist at Telenor Research and Development. His research has been focused on object-oriented database systems and highly available database systems. He received a Siv.ing. (MSc) in computer science in 1989 and a Dr.ing. (PhD) in computer science in 1993 from the Norwegian Institute of Technology.

Per Holager is a research scientist at SINTEF Communication and Telematics. He has almost 30 years of research experience within language design, compiler construction, software configuration management and database systems. He received a Siv.ing. (MSc) in technical physics in 1969 from the Norwegian Institute of Technology.