On recovery and consistency preservation in distributed real-time database systems

Sanny Gustavsson, Sten F. Andler
Högskolan i Skövde
{sanny,sten}@ida.his.se

Abstract. Consistency preservation of replicated data in distributed databases is nowadays a well-covered topic. Both pessimistic approaches (such as two-phase commit) and optimistic approaches (such as eventual consistency) have been thoroughly investigated. A current research trend is to move towards optimistic consistency preservation mechanisms, where immediate consistency at each transaction commit is traded off for increased predictability, availability and performance. In this paper, we consider the impact of optimistic consistency preservation techniques on the recovery process in distributed real-time database systems. We identify uninvestigated problems within this field, such as the absence of timely and optimistic recovery mechanisms, and our lack of understanding of the requirements to be met by applications operating in eventually consistent systems. We also suggest approaches for further research on these subjects.

1 Introduction

The distributed real-time database field of research is interesting for several reasons. Not only is the demand in industry for timely and dependable database systems high, the inherent complexity of distributed and concurrent systems coupled with the need for predictability in real-time systems also creates unique and interesting problems. The project described herein addresses the problem of recovering the contents of a crashed node in a distributed real-time database. We are particularly interested in recovery in eventually consistent databases, i.e., databases where availability, predictability and performance are improved by allowing replicas to be temporarily inconsistent. Such databases are interesting in that they must be convergent, which means that if all transactions in the systems are quiesced, all replicas must eventually become consistent. Eventually consistent databases also put restrictions on the applications that use the database, since they must be tolerant of stale data.

We argue that many systems today (e.g., banking systems and base stations for mobile telephony) must allow for temporary inconsistencies in order to provide a reasonable level of availability. These systems must, however, also be able to guarantee that a consistent state will eventually be reached.

In the following section we describe the project background and define the problem area. Section 3 lists the problems that we want to address, while section 4 suggests some possible approaches to solving these problems. Finally, section 5 contains our conclusions, including a discussion on future work and how our research fits in with other projects within the same field.

2 Background

A *distributed system* consists of several autonomous but interconnected processing elements (denoted *nodes*), which work together to achieve a common goal. Some important properties of distributed systems are:

- **Localized processing power**

  Computations can be performed at the site where they are needed, for example close to sensors and/or actuators, which means that less cabling is needed and that communication costs are decreased compared to a centralized system. This property is especially important in embedded systems.

- **High fault tolerance**

  In a distributed system, it is possible to replicate data and services, so that if a node goes down, the data and services of that node can still be available to the user on another node. This is an important property for systems that require high availability.

Many distributed systems contain some form of database. In a *distributed database*, the data is dispersed on several different nodes. Of particular interest to this paper is the concept of data replication. In a distributed database, what is seen by applications as a single, logical database object can in fact be stored as several physical copies, *replicas*, of that object. These replicas may be stored on different nodes to achieve a high degree of fault-tolerance and availability.

In a *real-time system*, correct behavior implies not only functional correctness, but also conformance to timeliness requirements, such as task deadlines and minimum delays. Many real-time systems are also *embedded* systems, which means that their primary function is not information processing (Burns & Wellings 1997). An example of such a system is a microprocessor-controlled washing machine.

A *distributed real-time system* must deal with not only the issues inherited from both these system classes, but also with several unique problems that exist in few other system types, such as predictable communication, multi-node load balancing, and timely consistency management.

2.1 Consistency preservation

As discussed in the previous section, a database object in a distributed database may be represented by several physical replicas on different nodes in the system. For the database to remain consistent, all such replicas must be kept identical and adhere to all constraints in the database specification. This means that any updates on the logical object must be reflected by all the physical replicas.

There are two main approaches to maintaining consistency in a distributed database – pessimistic and optimistic. We say that the pessimistic approaches support immediate consistency, while optimistic approaches only guarantee eventual consistency. These concepts are elaborated in the following two paragraphs.

- **Immediate consistency**

  Pessimistic techniques for consistency preservation ensure that all replicas of all logical database objects accessed by a transaction are consistent when that transaction commits, i.e., consistency is *immediately* achieved. There are several well-defined techniques for this, the most basic and well known of which is the *two-phase commit* protocol (Gray & Reuter 1993).

- **Eventual consistency**

  Optimistic consistency preservation techniques are based on the concept of *eventual* consistency. The idea is to trade off immediate consistency at each transaction commit for increased predictability, availability, and performance. This means that a transaction may commit updates to a local replica of a logical database object without propagating the updates to the nodes containing additional replicas of that object. Thus, the local performance and availability are increased, since all overhead associated with communication protocols and distributed commit protocols is removed. Examples of optimistic consistency preservation protocols are Distributed Optimistic Two-phase Locking (Carey & Livny 1991) and Lazy Replication (Breitbart & Korth 1997).

2.1.1 The consistency trade-off

Although a large research effort has been put into pessimistic consistency preservation techniques, the field is moving towards optimistic techniques for the performance benefits inherent in such techniques. We argue that in many of today's large-scale systems, temporary inconsistencies are inevitable. For example, although banking is a field where the demands on consistency could be assumed to be strict, extracting money from an ATM would often be impossible (due to, e.g., network partitions) if details of the transaction would have to be propagated to all nodes (i.e., bank offices) in the system before any money could be delivered. Thus, to increase the availability of the ATMs, the node at which the transaction was performed must be allowed to be temporarily inconsistent with the rest of the nodes in the system. In this type of system, eventual consistency is required. In the ATM system, for example, money must not be irrecoverably lost.

Since conflicts will occur in any eventually consistent system (e.g. the last $100 may be extracted from a given bank account simultaneously at two different nodes), conflict detection and resolution mechanisms must exist in the system. Also, applications must be aware of (and tolerate) the fact that they may be working with stale or inconsistent data. We say that applications must be convergent.

2.2 Database recovery

When a distributed database node crashes, its main-memory contents are lost, and must somehow be recovered. Traditionally, the most common method is to keep, on stable storage, a *log* of all updates performed on the node's data (Gray & Reuter 1993). When a node recovers, it reads and applies all the log entries to the database. Since the log can grow arbitrarily large, logging is nearly always accompanied by the taking of periodic *checkpoints*, i.e., complete database images written to stable storage. Checkpointing optimizes the time needed for recovery, since only the log records postdating

the most recent checkpoint have to be read during the recovery process.

2.2.1 Distributed recovery

In distributed databases, the consistency of the recovered data must also be considered. If immediate consistency is enforced, once the node is reintegrated in the system, its data must be consistent with the data on the other nodes. If only eventual consistency is required, the reintegrated node must still be in a state that is eventually consistent, i.e., a state that would become consistent after a bounded time period in a quiescent system.

Diskless distributed recovery
For some embedded systems, such as those working in environments with electromagnetic radiation or heavy vibrations, it may not be practical to use disks as permanent storage for checkpoints and logs. Also, many disk drivers are unpredictable, which makes them unsuitable for real-time systems. In addition, disks add to the cost of building the system. We thus see a need for recovery mechanisms that do not depend on disks.

In a replicated database, the inherent redundancy in the system can be used for recovery. Instead of reading a checkpoint from disk, the contents of another node in the system (the *recovery* source) can be copied to the recovering node (the *recovery target*). If updates are performed at the recovery source concurrently with the recovery process, those updates must also be transferred to the recovery target after the database image has been copied (Leifsson 1999).

3 Problem definition

This section gives a brief introduction to the aspects of diskless distributed recovery that we have chosen to focus on. For a more in-depth description of these topics, see Gustavsson (2000).

3.1 Timely recovery

For a recovery technique to be suitable for a real-time system, it must be timely. So far, few recovery algorithms prioritize or even consider timeliness. The only recovery algorithm designed for real-time systems that we have found (Song et al. 1999) does not consider consistency issues. We would like to examine methods for recovery in eventually consistent databases, focusing on their timeliness, i.e., whether they can be made predictable and sufficiently efficient for inclusion in a real-time system.

3.2 Consistency preservation in recovery

Any recovery mechanism in a distributed database must consider consistency preservation to ensure that the node is integrated correctly with the rest of the system once recovery is complete. The easiest way to ensure consistency is to quiesce the rest of the system while recovery is taking place. However, this may not be an option in systems that require high performance and availability.

If a diskless recovery mechanism (in section 2.2.1) is used, special care must be taken to preserve consistency, since the recovery source's view of the database may be updated as it is being sent to the recovery target. We wish to investigate how this affects the recovery process, especially in an eventually consistent system.

3.3 Application constraints

While eventual consistency improves several important attributes of a system, it also restricts the types of applications that may operate in the system. As discussed in section 2.1.1, applications in an eventually consistent system must be able to tolerate the fact that they may be working with stale or inconsistent data. We wish to establish a formal definition of the necessary properties of convergent applications.

4 Approaches

In this section, we present some approaches that we plan to take in tackling the problems of the previous section.

4.1 Creating a taxonomy

Initially, we aim to formulate a taxonomy for distributed recovery in different kinds of systems. For example, recovery (especially diskless recovery) in a quiescent system differs significantly from recovery in a non-quiescent, eventually consistent database. We need to distinguish different recovery scenarios by identifying the system properties that affect the recovery mechanism.

4.2 Modifying existing techniques

We intend to examine already existing recovery techniques to see whether they can be refined for use in a diskless and eventually consistent real-time database system. Some initial work in this direction has already been performed (Leifsson 1999, Gustavsson 2000). For example, according to Leifsson (1999), sending the data from a recovery source to the recovery target in diskless recovery is

analogous to taking a fuzzy checkpoint (Li et al. 1995).

4.3 Analysis and theoretical proofs

To add weight to any claims that we make about a recovery algorithm's timeliness and/or non-interference with conflict detection and conflict resolution protocols, we need to logically prove that it has those properties. For us to be able to perform the required analysis and proofs, we need a way of formally reasoning about these subjects. Finding suitable methods to do this, or, if required, devising new ones will be an important part of our project.

4.4 DeeDS integration

Finally, we wish to verify the feasibility of the diskless recovery algorithm by implementing it in a real-world system and running benchmark tests. DeeDS, a distributed real-time database system prototype developed by the distributed real-time systems research group at university of Skövde (Andler et al., 1996), is well suited for such an implementation.

5 Conclusions

This project is in its early stages. We have completed an initial literature survey and analysis (Gustavsson 2000) and are currently investigating suitable methods. In the remainder of this section, we briefly discuss our future work and how it relates to existing research in this field.

5.1 Future work

The research problems that we wish to investigate are in three distinct categories:

- General problems in distributed recovery
  - Finding timely recovery algorithms
  - Exploring the relationship between consistency preservation and recovery
- Eventual consistency problems
  - Devising a recovery method for eventually consistent databases that does not interfere with consistency preservation
  - Defining constraints on applications that operate in an eventually consistent database
- Diskless recovery extensions
  - Relaxing some of the assumptions made by Leifsson (1999), such as allowing multiple node failures or incremental recovery
  - Performing benchmark test of a real-world implementation of the diskless recovery algorithm

5.2 Related work

Work exists on optimistic consistency preservation approaches that are similar to our notion of eventual consistency. For example, the independent update algorithm briefly described by Ceri et al. (1994) shares many aspects with eventual consistency. Unfortunately, little is said about the implications that using the independent update algorithm has on the database system and its applications. The only things mentioned by the authors are that the reconciliation (conflict resolution) algorithm should ensure one-copy serializability, and that the applications must be capable of handling stale data.

Concurrency control, which is similar to consistency preservation, is also a well-covered area. However, we have been able to find little or no such research that focuses on the interaction between these techniques and recovery.

We also argue that diskless recovery is a fresh and interesting area that requires further exploration. Also, there is, to our knowledge, no research on application requirements for eventually consistent databases.

5.3 Contributions

Our work focuses on aspects of distributed recovery that have, so far, received very little attention. We look at timely recovery, which should be increasingly important as the number of systems with demands on timeliness continues to grow. Similarly, many embedded systems operate in hostile environments, where the use of disks may not be an option. In such systems, a diskless recovery mechanism may eliminate the need for stable storage, while decreasing the cost of the system. Finally, the research community as a whole is moving towards optimistic approaches for consistency preservation, as we realize that in many real-world systems, immediate consistency cannot be enforced while meeting the high demands on performance and availability. Therefore, node recovery in an eventually consistent database is a relevant and interesting topic.

Acknowledgments

We extend our thanks to Jonas Mellin and Robert Nilsson, who were both able to give us feedback on short notice.

References

Andler, S., Hansson, J., Eriksson, J., Mellin, J., Berndtsson, M. & Eftring, B. (1996), DeeDS towards a distributed and active real-time database system, SIGMOD Record 25(1), 38-40

Breitbart, Y. & Korth, H. (1997), Replication and consistency: Being lazy helps sometimes, in Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, Tucson, Arizona.

Burns, A. & Wellings, A. (1997), Real-Time Systems and Programming Languages, Addison-Wesley.

Carey, M & Livny, M. (1991), Conflict detection tradeoffs for replicated data, ACM Transactions on Database Systems 16(4), 703-746.

Ceri, S., Houtsma, M., Keller, A. & Samarati, P. (1994), A classification of update methods for replicated databases, Technical Report CS-TR-91-1392, Stanford University, Computer Science Department.

Gray, J. & Reuter, A. (1993), Transaction Processing: Concepts and Techniques, Morgan Kaufmann, chapter 10.

Gustavsson, S. (2000), On recovery and consistency preservation in distributed real-time database systems (HS-IDA-MD-00-015), Master's thesis, University of Skövde.

Li, X., Eich, M., Joseph, V., Gulzar, Z., Corti, C., Nascimento, M. & Peltier, A. (1995), Checkpointing and recovery in partitioned main memory databases, in Proceedings of the International Conference on Intelligent Information Management Systems, Washington, DC, June 7-9, pp. 59-63.

Leifsson, Æ. (1999), Recovery in distributed real-time database systems (HS-IDA-MD-99-009), Master's thesis, University of Skövde.

Song, E-M., Kim, Y-K., Ryu, C., Choi, M., Kim, Y-K., Jin, S-I., Han, M-K. & Choi, W. (1999), No-log recovery mechanism using stable memory for real-time main memory database systems, in Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications.