Analyzing Mobile Transactions Support for DBMS

Patricia Serrano-Alvarado*, Claudia L. Roncancio, Michel Adiba
LSR-IMAG Laboratory
BP 72, 38402 St-Martin d'Hères, France
e-mail: Firstname.Lastname@imag.fr

Abstract

This paper focuses on transaction management in mobile environments. Mobility introduces new dimensions such as movement and disconnection that affect transaction management. Several proposals concerning mobile transactions have been done, but a detailed comparative analysis does not exist. In this paper we deeply analyze and compare how mobility features have influenced the ACID properties and the execution model of some proposals. Furthermore, based on our analysis, we introduce a sketch of the design of a Mobile Transaction Service.

Keywords: Mobile transactions, databases, mobility, ACID properties, commit, DBMS

1. Introduction

Data management in mobile environments is gaining a lot of attention today with the emergence of mobile computing. Nevertheless, new kind of problems are generated due to the nature of mobile clients. To that extent, database techniques (system architectures [10], caching, query processing, etc [14]) should be revisited. In our research group, we are interested in developing a DBMS in a networking environment by defining, designing and implementing it as a set of services that should be flexible enough to deal with a large spectrum of architectures/applications (NODS Project [3]). In such a context, transaction management is a key issue. Our interest is the conception and experimentation of a Mobile Transaction Service (MTS) which will provide transaction support suitable for mobile environments. To achieve this goal, the first step is to make a deep analysis and comparison of previous proposals. The literature on the subject is important and some attempts to analyze proposed models have been made [6][11]. However, we think that it is necessary to make an extensive analytical comparison of these models.

*Supported by the CONACyT scholarship program of the Mexican Government

We are considering a mobile computing environment with a network consisting of stationary and mobile hosts (SH, MH). Shared data are distributed over several database (DB) servers running on SH. MH could be of different nature ranging from PDA to personal computers. An MH changes its location and network connections while transactions are being processed. While in motion, an MH retains its network connections through the support of SH which act as Base Stations (BS).

In this paper, we analyze and compare mobile transaction proposals with an overview in Section 2 and an analysis in 3. Moreover, in Section 4 we present preliminaries ideas for the construction of an MTS. In Section 5 we conclude the paper.

2. Mobile transactions

Informally, *a transaction is a set of operations that translate a database from a consistent state into another consistent state*. Transaction managers (TM) offer ACID properties by implementing commit protocols, obtaining serializable executions, controlling visibility of non-committed transactions, supporting recovery, etc. Nevertheless, very often, ACID properties are not appropriate and several models relaxing these properties have been proposed [7]

In the context of mobile computing, there exist several interpretations of mobile transactions (MT). For us, *an MT is a transaction where at least one mobile host takes part in its execution*. The participation of an MH introduces dimensions inherent to mobility such as: movement, disconnections and variations on the quality of communication. As we will see in the following, TM supporting MT have to adapt their functionalities to deal with these dimensions.

In the scope of this paper we focus on systems with a client-server architecture where clients are MH having stockage/processing capabilities and where the server is on the wired network. The server provides resources and transaction management. In the following, we consider the system in *connected mode* if the MH and the server can establish communication if necessary; otherwise it is in *discon-*

nected mode. Whenever we use the term "local", as in local transactions and local processing, we refer to MH.

2.1. Proposals overview

The remainder of this section reviews some of the work done in MT. There are also, interesting works such as [6] and [17] which propose solutions concerning movement and disconnections but not transaction properties. Due to space limitations we do not include their analysis in this paper. They are included in [15] as well as summary tables of the analysis presented here.

Clustering proposal [13] assumes a fully distributed system and is designed to maintain DB consistency. The DB is dynamically divided into clusters, each one groups together semantically related or closely located data. A cluster may be distributed on several strongly connected hosts. When an MH is disconnected it becomes a cluster by itself. For every object two copies are maintained, one of them (*strict version*) must be globally consistent, and the other (*weak version*) can tolerate some degree of inconsistency but must be locally consistent. MT are either *strict* or *weak*. Weak transactions access only weak versions whereas strict ones access strict versions.

Two-tier replication [9] considers both transaction and replication approaches for mobile environments where MH are occasionally connected. A master version for each data and several replicated versions (copies) exist. Two types of transactions are supported: *base* and *tentative transactions*. Base transactions are executed accessing master versions (lazy-master replication scheme) whereas tentative transactions are executed accessing tentative versions (local copies). Tentative transactions may perform updates on the MH in a disconnected mode.

Pro-motion [16] is an MT processing system that supports disconnected mode. *Compacts* are introduced to allow local executions on MH. Necessary information to manage the compact is encapsulated in it. To improve autonomy and to increase concurrency, objects semantics are used in the construction of compacts whenever possible. Compacts are the basic unit of caching and control.

Reporting [1] analyzes nested transactions and open-nested transactions (sagas, split and multitransactions) showing their limitations for mobile environments. [1] considers a mobile database environment as a special multi-database system with specific requirements, where transactions on MH are considered as a set of subtransactions. They propose an open-nested transaction model that supports *atomic*, *non-compensatable transactions* and two additional types: *reporting* and *co-transactions*. While in execution, transactions (executed on the MH) can share their partial results and partially maintain the state of a mobile subtransaction on a BS.

Semantics-based [2] focuses on the use of object semantics information to improve autonomy of MH in disconnected mode. This contribution concentrates on *object fragmentation* as a solution to concurrent operations and to limitations of MH storage capacity. This approach uses objects organization and application semantics to split large and complex data into smaller manageable fragments of the same type. Each fragment can be cached independently and manipulated asynchronously. Fragmentable objects can be aggregate items, sets, stacks and queues.

Prewrite [12] tries to increase data availability on MH by introducing a *prewrite* operation in addition to standard writes. A prewrite makes data value visible at *precommit* before the commit of the MT. Permanent updates on the DB are performed later by the write operation at commitment. Two variants of the data are maintained: the *prewrite* and the *write*. Prewrite variant reflects future state of data but may be structurally slightly different from the corresponding write value e.g., in an object of type document the prewrite is an abstract and the write is the complete document.

3. Analysis of transaction proposals

Our analysis is based upon five characteristics: execution model, atomicity, consistency, isolation and durability. We consider that to characterize functionalities related with transactions, it is essential to know how transactions deal with ACID properties and how they are executed. In Section 3.1 we analyze each proposal to identify their transaction type and the way to execute it. In 3.2, 3.3, 3.4 and 3.5 we compare the models and identify common points regarding ACID properties.

3.1. Execution model

In **Clustering**, strict transactions are executed when hosts are strongly connected and weak transactions when MH are disconnected. Two kinds of operations are introduced *weak reads* and *weak writes*. Strict transactions contain standard reads and writes, whereas weak transactions contain weak operations. When reconnection is possible (or when application consistency requires it) a synchronization process, executed on the DB server, allows the DB to be globally consistent.

Two-tier replication uses two types of atomic transactions (base and tentative) that differ in the version of data they access. After a disconnected execution, the BS will re-execute tentative transactions as base transactions to reach global consistency. This re-execution is the way to make local updates persistent.

Pro-motion considers the entire mobile system as one extremely large long-lived transaction executed on the server. The management of compacts is performed by a *compact manager* at the DB server, a *compact agent* at the MH and a *mobility manager* at the BS. The *compact manager* constructs the compacts, it will act as a front-end for the DB server and appears to be an ordinary DB client executing a single, large long-lived transaction. On each MH, the *compact agent* is responsible for cache management as well as for transaction processing, concurrency control, logging and recovery. The *mobility manager* is in charge of transmissions between the agents. MH transactions are executed locally even in connected mode. A synchronization process is executed by the compact agent and the compact manager at reconnection.

In **Reporting**, an MT is structured as a set of transactions, some of which are executed on the MH. *Atomic transactions* have standard abort and commit properties. *Non-compensatable transactions* at commit time delegate to their parent all operations they have invoked. *Reporting transactions* report to another transaction some of their results at any point during execution. A report can be considered as a *delegation of state* between transactions. *Co-transactions* are reporting transactions where control is passed from the reporting transaction to the one that receives the report. Co-transactions are suspended at the time of delegation and they resume their execution when they receive a report.

In **Semantics-based**, no assumptions are made about the transaction structure. MH fragment request includes two parameters: *selection criteria* and *consistency conditions*. The selection criteria indicates data to be cached on the MH and the required fragment size. The consistency conditions specify constraints to preserve consistency on the entire data. Data fragmentation executed on the sever allows fine-grain concurrency control. Exclusive master copies of fragments are given to the MH and transactions can be entirely executed on it. A reconciliation process is executed by the server when reconnection occurs. This model may be used with different transaction types.

In **Prewrite**, the main idea is to divide the transaction execution between the MH and the DB server. Three operations (*prereads, prewrites* and *precommit*) that will be executed by the TM are proposed. Ordinary reads and permanent writes are made by a data manager (DM) at the DB server. The BS has logging capacities and maintains close relationship with the DM. The transaction execution is divided in two parts, first, the TM executes local transactions that finish with a precommit. In the second part, the DM makes prewrites permanent and commits the MT. This model considers that MT are long-lived and implementation can be made with nested and split transactions.

All proposals but Reporting assume that MT are requested from MH. In Reporting, transactions can be requested by any host.

3.2. Atomicity

Except for Reporting and Semantics-based, transaction validation is done in two steps. The first one is realized on MH (*local commit*) and the second one (*commit*) at the BS/DB server. Clustering, Two-tier replication, Pro-motion and Prewrite execute local commit, each one with specific characteristics:

- Clustering and Two-tier replication make local commit only in disconnected mode using special transaction types. In connected mode an atomic commit protocol is used (e.g., two phase commit) and it includes participation of several clusters/hosts.
- Pro-motion and Prewrite do not differentiate connected and disconnected mode. Local commit is performed using an atomic commit protocol.

At the second step of the validation process, locally committed transactions execute commit to make updates permanent on the DB server. Transaction commitment can involve reconciliation mechanisms or transactions re-execution.

- Reconciliation in Clustering is made syntactically where weak transactions are aborted or rolled back if their weak writes conflict with strict transactions.
- In Two-tier replication, if base transactions fail, even by taking into account the *acceptance criteria* (attached to each tentative transaction), then the tentative transactions are aborted.
- In Pro-motion, compacts involved in locally committed transactions are checked. If some compacts are no more valid, then MT are aborted and a *contingency procedure* (attached to each local commit) is executed to obtain semantic atomicity.
- In Prewrite, neither reconciliation nor re-execution are made. By the transaction processing algorithm and locking protocol, Prewrite ensures that locally committed transactions will commit at the DB server.

The approach is different in Reporting where each subtransaction is atomic but this does not prove the atomicity of the global MT. With *atomic transactions* compensatable transactions are assumed so (semantic) atomicity is guaranteed. In *Non-compensatable transactions*, *reporting* and *co-transaction* delegation does not affect atomicity because it does not require the invoking transaction of an operation to be the transaction who either commits or aborts the operation.

In Semantics-based, transactions are considered long-lived. As MH are responsible of local transactions commit it would be possible to support atomic or non-atomic transactions.

Conceptually, Semantics-based, Pro-motion, Prewrite and Reporting consider transactions as long-lived ones. If these transactions are executed on multidatabase systems, global atomicity depends on the autonomy of each DBMS.

Cascading aborts may occur in Clustering, Two-tier replication and Pro-motion. Local committed transactions modify local data, consequently only aborts of local transactions are provoked. These aborts just concern weak and tentative transactions because local results are exclusively available for these types of transactions.

3.3. Consistency

Clustering and Two-tier replication maintain consistency of replicated data with two versions. Both versions are located on the MH, one of them (weak/tentative) is used to support data evolution in disconnected mode. The second one (strict/master) must be always consistent but sometimes it will contain old versions (in disconnected mode). Consistency in strict/master versions is preserved using one-copy serializability methods (e.g., quorum consensus, master copy). Some particularities are:

- In Clustering, semantic information is used to specify the degree of inconsistency for weak versions. This degree may be bounded limiting the number of local commits, the number of transactions that can operate on inconsistent copies, the number of copies that can diverge, etc. There exist also a *function h* that controls this degree by projecting strict operations on weak versions. Full consistency is achieved by merging different copies of the same data located at different clusters.
- In Two-tier replication, tentative data versions are discarded at reconnection since they are completely refreshed from master versions.

It seems to us that weak/tentative transactions have drawbacks with respect to strict/base transactions, in the resynchronization process (reconciliation in Clustering and re-execution in Two-tier replication).

Pro-motion and Semantics-based exploit semantic information to construct compacts and fragments:

- For Pro-motion the *compact* represents an agreement between the DB server and the MH. The compact manager and the DB server encapsulate in compacts: *data, type specific methods, state information, consistency rules,* and *obligations*. If the compact agent and compact manager respect all these conditions, the use of compacts will not affect DB consistency. The compact designer can determine correctness criteria and concurrency control methods per compact.
- In Semantics-based, to preserve consistency, objects must carefully support *split* (to make fragments) and *merge* (to reconciliate fragments) operations. Another restriction to preserve consistency is to provide *consistency conditions* (supplied by applications) on the entire object.

In Reporting, new ways to achieve consistency are not proposed, but subtransactions can be related to compensating transactions in order to maintain semantic consistency in case of abortions.

Prewrite assures that the transaction processing algorithm along with the lock-based protocol, produce only serializable histories. This serializability is based on the pre-commit order of MT.

It is important to notice that semantic information of objects is essential to guarantee consistency in mobile applications. All analyzed models exploit object semantics in different ways. Clustering defines degrees of inconsistency based on the application semantics. Two-tier replication manages an acceptance criteria between tentative and base transactions. Pro-motion uses semantic information to construct compacts and Semantics-based to split objects. Reporting makes delegation based on semantic requirements, and Prewrite defines semantically identical data variants (prewrite/write objects).

3.4. Isolation

Isolation is not strictly enforced by all proposals, some of them allow *visibility* of intermediate transaction results.

Clustering, Two-tier replication, Pro-motion and Semantics-based give visibility of local committed results to local transactions on the same MH. On the other hand, Prewrite at local commit makes the results public to all hosts. In Reporting, visibility is permitted in atomic, reporting and co-transactions but not in non-compensatable transactions. An atomic transaction can commit its execution even before the commit of its parent, and its modifications to the DB become visible for others transactions. In reporting and co-transactions the objective is precisely to allow visibility of partial results while in execution.

Taking Pro-motion and Reporting as open-nested transactions, global isolation is not respected since subtransactions are not executed isolately. After the synchronization process, Pro-motion splits its long-lived transaction. All operations that have been successfully synchronized form a separate transaction that is committed on the DB server. Results of this split (committed) transaction will be visible for all the DB environment.

To manage isolation (restraint visibility) Clustering and Prewrite propose new conflict solution tables.

- Clustering uses strict two phase locking and proposes four lock types that correspond to weak and strict operations (WR, WW, SR, SW). Four conflict tables for lock compatibility are proposed. The projecting *function h* utilizes conflict tables to reflect strict operations on weak versions depending on the application consistency requirements. For example, strict consistency requires translating a strict write on an object into strict writes on all its copies (strict and weak ones). Consequently, a SW lock is non compatible with any other lock. Weak transactions release their locks at local commit and strict transactions at commit.

- As Clustering, Prewrite uses a two phase locking protocol and the conflict operation table includes preread and prewrite operations (PR, PW, R, W). As prewrite and preread locks are managed at TM level and read and write locks at DM level, there exist no conflict between prewrite/preread and write/read locks. To make prewrites permanent the prewrite lock must be converted into a write lock so that the DM can write and commit the MT. Preread locks are released at local commit time whereas prewrite/write/read locks at commit time.

In our opinion, Prewrite approach is interesting in applications using objects that can have two variants (write/prewrite value) as *design objects* (the prewrite represents a model of the design) or *document objects*. In these object types prewrites are different from writes and availability is improved with two variations of the "same object". Otherwise, using simple objects, prewrites are identical to writes and the algorithm behaves as using relaxed two phase locking.

Since in Pro-motion the compact designer can determine correctness criteria and concurrency control methods per compact, they propose to use a ten level scale. Level 9 represents a serial execution of transactions and level 8 a serializable execution. Each succeeding level represents a lesser degree of isolation. At level 0 there are not isolation guarantees.

In Semantics-based, to ensure serializability, local transactions have access to cached fragments by conventional concurrency control protocols e.g. two phase locking.

3.5. Durability

Clustering, Two-tier replication and Pro-motion cannot guarantee durability before commit. Pro-motion with compacts can give some guarantees of durability, but may exist conditions that could not be respected because of disconnections e.g., there is a deadline (in the compact) that could not be reached, consequently, durability is hard to obtain in the synchronization process. In Reporting, subtransactions are durable if the parent transaction commits. Semantics-based and Prewrite models guarantee durability since local commit. The first one reduces fragments availability because it can hold fragments by an undefined period of time. The second one uses many messages exchanges to get locks from the BS. In the Prewrite algorithm, if an MT makes a local commit, it is sure to commit, Prewrite does not permit a local committed transaction to abort.

Note that logging issues are not discussed here. It seems that in the proposals we have analyzed these issues are not clearly studied, but we are currently investigating these aspects.

4. Towards a Mobile Transaction Service

For defining future DBMS, the NODS project develop an open, adaptable architecture that can be extended and customized on a per-application basis [3]. Our approach is characterized by a service oriented view of DB functionality [5]. All DBMS and related tasks are unbundled into services and applications use services as needed. In the design of services, particular attention is payed on their adaptability.

The Mobile Transaction Service (MTS), we are working on, will provide support to MT and will cooperate with other services such as the replication, persistence [8] and event services [4]. This section discusses some important issues about the MTS definition.

Overall functionalities As we have seen in previous sections, MT support includes standard TM functions, extra functions and particular implementations. The most important and new feature the MTS must support is disconnection management. Disconnections make difficult or impossible to implement traditional protocols that deal with ACID transactions. Concerning function implementation, we have already identified two important points that have to be modified because of disconnections: transaction validation process and consistency management. We consider that it is crucial to perform transaction validation in two steps that corresponds to *local commit* and *commit* introduced in 3.2. Disconnections make consistency management more complicated. It is necessary to adopt particular concurrency control protocols and synchronization process to offer some kind of serializability.

Transaction processing scenarios Our working hypothesis is to adopt an execution model flexible enough to satisfy different environments. The execution of MT may be performed in accordance with one of the following scenarios:

1. MT is entirely executed on the DB server.
2. MT processing is distributed between the MH and the DB server.
3. MT processing is distributed between several MH.
4. MT is executed on the MH.

Each one of these execution strategies has special characteristics and demands particular capabilities. In (1), the execution can be "traditional" but the MTS should be aware of the MH connectivity state to deliver results.

In (2) and (3), MTS must be able to perform distributed executions where participants could not communicate during executions. In this scenario, a two steps validation process would be appropriate. Further, consistency must be guaranteed with special concurrency control protocols and synchronization methods. In both scenarios the transaction coordination must be located at the DB server. For (3) this decision is due to the lack of predictability in the connection of MH at the same time.

In (4), MTS should ensure global consistency comprising MH updates. The MH has some freedom to manage the data locally but updates have to be incorporated in the DB server. As in (2) and (3), concurrency control and synchronization methods must be adapted.

Transaction model In our opinion, the support of one single transaction type is not enough for mobile environments. Different transaction types are needed depending on the execution strategy. For example we could use for (1) flat transactions, for (2) and (3) open-nested transactions and for (4) long-lived transactions. Open-nested transactions by their structure can support some kind of local commit (allowing data evolution, application blocking is reduced) and parallel processing (when execution is distributed between the MH and the MTS). Long-lived transactions are ad-hoc for the third execution strategy because of undefined disconnection time. The long-live transaction could be a simple transaction or an open-nested one.

Consistency and Durability We emphasize the importance of avoiding application blocking at MH in disconnected mode. To achieve this goal, local availability of consistent objects is necessary. As we have noticed, semantic approaches are well adapted to manage consistency in mobile context. Moreover, local commit is necessary to obtain visibility of transaction results that are not already committed at the DB server (in disconnected mode). Another important property to consider is durability of MT. Frequently, at resynchronization time, local committed MT have lower priority than non-MT; for mobile applications this is a great disadvantage. It is important to remark, that due to all changes introduced by mobility, also logging has to be adapted. The importance of logging increases because in addition to recovery purposes it is utilized to perform synchronization process.

Architecture The analyzed models showed that BS can be a significant support for the MTS. Besides establishing connection with MH, the BS can have server capabilities as logging, data caching, resynchronization process, concurrency control mechanisms, etc. Delegating functionalities to the BS allows the MTS to save communication costs and to improve response time. Consequently, for the MTS we will consider a three-tier architecture as client/agent/server. The client is on the MH and it may have stockage/processing capabilities. An agent on BS establishes connection between the client and the server and gives some kind of support to the server located on SH. This architecture and the specification of a prototype environment are part of our future work.

5. Conclusions

In this paper we analyzed different proposals that deal with mobile transactions. We compared them by discussing the execution model and the way the ACID properties are insured. We identified common trends on these proposals by trying to overcome the differences on terminology used in the literature. Finally, we discussed the design of a general mobile transaction service which is the subject of our ongoing research.

References

[1] P. K. Chrysanthis. Transaction Processing in a Mobile Computing Environment. In *Workshop on Advances in Parallel and Distributed Systems*, pages 77–82. IEEE, October 1993.

[2] P. K. Chrysanthis. Supporting Semantics Based Transaction Processing in Mobile Database Applications. *14th IEEE Symp on Reliable Distributed Systems*, September 1995.

[3] C. Collet. The NODS project : Networked Open Database Services. *ECOOP*, June 2000.

[4] C. Collet, G. Vargas-Solar, and H. Grazziotin-Ribeiro. Open Active Services for Data-Intensive Distributed Applications. In *IDEAS*, Yokahama-Japan, September 2000.

[5] K. R. Ditrich and A. Geppert. *Component Database Systems*. Morgan Kaufmann Publishers, 2001.

[6] M. H. Dunham and A. Helal. A Mobile Transaction Model that Captures Both the Data and the Movement Behavior. *ACM/Baltzer Journal on special topics in mobile networks and applications*, 2:149–162, 1997.

[7] A. K. Elmagarmid. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, 1992.

[8] L. García-Bañuelos and C. Collet. Towards an Adaptable Persistence Service: The NODS Approach. *TOOLS 2001 Workshop on Object-Oriented Databases*, March 2001.

[9] J. N. Gray, P. Helland, P.O'Neil, and D. Shasha. The Dangers of Replication and a Solution. In *Conference on Management of Data*, pages 173–182, Canada, June 1996.

[10] J. Jing, A. Helal, and A. Elmagarmid. Client-Server Computing in Mobile Environments. *ACM Computing Surveys*, 31(2), June 1999.

[11] S. K. Madria. Transaction Models for Mobile Computing. *15th IEEE International Conference on Distributed Computing Systems*, June 1995.

[12] S. K. Madria and B. Bhargava. A Transaction Model for Improving Data Availability in Mobile Computing. *To appear in Distributed and Parallel Databases*, 2001.

[13] E. Pitoura and B. Bhargava. Data Consistency in Intermittently Connected Distributed Systems. In *Transactions on Knowledge and Data Engineering*, November 1999.

[14] E. Pitoura and G. Samaras. *Data Management for Mobile Computing*. Kluwer Academic Publishers, 1998.

[15] P. Serrano-Alvarado, C. Roncancio, and M. Adiba. Mobile Transaction Supports for DBMS: An Overview. Technical Report RR 1039-I-LSR 16, LSR-IMAG, France, July 2001.

[16] G. D. Walborn and P. K. Chrysanthis. Transaction Processing in PRO-MOTION. In *14th ACM Annual Symposium on Applied Computing*, San Antonio Tx, February 1999.

[17] L. H. Yeo and A. Zaslavsky. Submission of transactions from mobile workstations in a cooperative multidatabase processing environment. In *Conference on Distributed Computing Systems*, June 1994.