Mobile Transaction Management in Mobisnap*

Nuno Preguiça[1], Carlos Baquero[2], J. Legatheaux Martins[1], Francisco Moura[2], Henrique Domingos[1], Rui Oliveira[2], J. Orlando Pereira[2], and Sérgio Duarte[1]

[1] Departamento de Informática, FCT, Universidade Nova de Lisboa, Quinta da Torre, 2845 Monte da Caparica, Portugal, {nmp,jalm,hj,smd}@di.fct.unl.pt

[2] Departamento de Informática, Universidade do Minho, Largo do Paço, 4700 Braga, Portugal, {cbm,fsm,rco,jop}@di.uminho.pt

Abstract. In this paper we describe a transaction management system designed to face the inherent characteristics of mobile environments. Mobile clients cache subsets of the database state and allow disconnected users to perform transactions independently. Transactions are specified as mobile transactional programs that are propagated and executed in the server, thus allowing the validation of transactions based on application-specific semantics. In the proposed model (as in others previously presented in literature) the final result of a transaction is only determined when the transaction is processed in the central server. Users may be notified of the results of their transactions using system support (even when they are no longer using the same application or even the same computer). Additionally, the system implements a reservation mechanism in order to guarantee the results of transactions performed in disconnected computers.

1 Introduction

Database systems designed for mobile computing must handle the inherent characteristics of those environments [10]. In particular, mobile applications have to face periods of disconnection that may arise due to economical factors, unavailable connectivity or application model. To allow mobile users to continue their work even in these periods, it is common to rely on optimistic replication techniques. In such approaches, shared data is replicated on mobile computers and users are allowed to continue their work while disconnected. Updates performed by disconnected users are logged and later propagated to servers. Several problems arise from such approaches: (1) updates performed by different disconnected users may conflict among them; (2) due to the previous problem, it is usually impossible to determine the result of an update in the mobile device; (3) mobile users may wish to perform operations over data that is not locally replicated. In this paper we present the Mobisnap [6] approach to tackle these problems in a relational database system.

* This work was supported in part by Praxis XXI

The Mobisnap system is based on a central database server that holds the primary replica of all data items. Mobile clients replicate subsets of the database information and mobile users are allowed to update database information through the submission of "mobile transactions". These "mobile transactions" are specified in an extended subset of the PL/SQL language [8], allowing programmers to clearly state the intended semantics of each operation — pre-conditions, post-conditions and different alternatives may be defined for each transaction. The final result of a "mobile transaction" is only determined when the transaction is performed in the central database. The Mobisnap system provides linguistic and system support to allow mobile users to be notified of this final result (even when they are no longer using the same application or even the same computer).

"Mobile transactions" submitted while disconnected are tentatively applied to the local database state. The result of this local execution represents the expected final result of the transaction. However, concurrent updates performed by other users may lead to a different result when the transaction is performed in the master database. To alleviate this problem, we have designed a reservation mechanism that allows mobile clients to make reservations upon database information. Therefore, mobile clients are able to determine the result of mobile transactions that only depend on reserved information. This mechanism combines leasing [1] with an extension of, previously proposed, escrow techniques [5].

The remainder of this paper is organized as follows: Section 2 discusses the motivation and design principles; Section 3 describes the Mobisnap transactional model; Section 4 discusses related work and Section 5 concludes the paper with some final remarks.

2 Motivation and Design Principles

In this section we present the ideas that lead to the Mobisnap approach to mobile transaction processing. We illustrate the proposed mechanisms with a system intended to support salespeople. The system manages not only information about the products but also the personal datebooks of sellers where demonstrations can be scheduled by the salesman or by the sales department. Although the presented database is very simplified, we believe that this example illustrates the proposed ideas. Furthermore, similar problems can be found in different applications for mobile environments. Details about the outlined mechanisms will be described in the next section.

2.1 Mobile transactions should be conceptually simple

In mobile computing, it is often necessary to rely on optimistic replication techniques to face disconnection: transactions are tentatively executed in mobile units and they are later validated and integrated in the master database. In Mobisnap, mobile transactions are defined in an imperative language based on PL/SQL [8], thus allowing programmers to specify the intended semantics for

```
------------------------------------ NEW ORDER ------------------------------------
BEGIN
  SELECT price, stock INTO prd_price, prd_cnt FROM products WHERE name = 'BLUE THING';
  IF prd_price <= 10.00 AND prd_cnt >= 50 THEN
    -- update orders, current stock, ...
    NOTIFY( 'SMTP', 'sal-07@thingco.pt', 'Order completed ...');
    COMMIT;
  ENDIF;
  ROLLBACK;
ON ROLLBACK NOTIFY( 'SMS', '351927435456', 'Impossible order ...');
END;
------------------------------------ NEW DEMO ------------------------------------
BEGIN
  SELECT count(*) INTO cnt FROM demo WHERE day='17-FEB-2000' AND hour=10;
  IF (cnt = 0) THEN
    -- update demos, send notification if appropriate, ...
    COMMIT;
  ENDIF;
  SELECT count(*) INTO cnt FROM demo WHERE day='18-FEB-2000' AND hour=9;
  IF (cnt = 0) THEN
    -- update demos, send notification if appropriate, ...
    COMMIT;
  END IF;
  ROLLBACK;
ON ROLLBACK NOTIFY( 'SMS', '351927435456', 'Impossible demo ... ');
END;
```

Fig. 1. Definition of two mobile transactions (declaration of variables is ommitted).

each transaction, testing pre and post-conditions and defining possible alternatives. In the example of Figure 1 we present two mobile transactions. In the "new order" transaction the precise preconditions are tested — the customer wants to order some product if the price is less than a given value and there are enough products in stock. There is no need that the values in the server are the same that have been seen in the mobile unit. In the "new demo" tranaction two alternative schedules are checked for the request of a new demo. These mobile transaction should be instanciated from predefined templates using the values selected by the users.

The result of a mobile transaction is completely and safely determined by the execution of the transaction program in the server. Instead of integrating mobile updates in the server relying on read/write and write/write conflicts to validate transactions, this approach allows the use of semantic information associated with the operations performed. Therefore, conflict detection and resolution can be specified in a precise and simple way in the code of the mobile transaction. The programmer can reason about the mobile transaction as a mobile program sent from the client to the server holding the primary copy and executing later on that server. We believe that this approach offers a conceptually simple model, allowing simple and powerfull operation definition. We also believe that this model allows a high degree of concurrency and scalability since the asynchronous nature of the client/server interaction and the semantics of mobile transactions only require short lasting locks in the database (no "long transaction" processing is required).

2.2 Awareness should be a first-class citizen

The common client/server approach to transaction processing assumes that the user that issues a transaction is connected to the system when its execution is completed. Therefore, users can be immediately notified of the results of their transactions and may perform alternative actions if necessary — a typical example is the flight reservation system. The proposed mobile transaction model differs from this approach in a fundamental way: usually, users will not be connected to the system when the results of their transactions are determined. Therefore, the propagation of transactions' results to the client machines is not sufficient (and it will be sometimes impossible due to the disconnection of those devices). The system should integrate a simple and clean mechanism that allows the active notification of users, to be used when it is appropriate, using the users preferred transport mechanisms – electronic mail, SMS/pager messages, ... (additionally, a pull-based mechanism should also be provided). In the examples of Figure 1 it is possible to observe the use of this mechanism in different situations and using alternative transports depending on the importance of the messages.

2.3 Guarantees are valuable

In the proposed mobile transaction model, as in others previously proposed in literature [11, 4, 2, 9], the result of a transaction submitted in a mobile unit is only determined when the transaction is finally executed in the database server. However, the transaction is tentatively performed in the mobile unit to provide a hint of its final result. In some applications, the ability to provide a stronger hint about the results of transactions would be very valuable — for example, salespersons would like to immediately guarantee that they could meet customers requests. To this end, we have integrated a reservation mechanism in the Mobisnap system.

This reservation mechanism combines and extends ideas used previously in [5] (escrow techniques) and [1] (leases). We have defined four types of reservations:

Escrow It is used to divide a partitionable resource. For example, different subsets of the available instances of a given product can be reserved by different salespersons.

Slot It is used to reserve the right to insert a record with pre-defined values. For example, someone may want to reserve the right to schedule a meeting in a room in a defined period.

Value-change It is used to reserve the right to change some values in the database. For example, someone may reserve the right to change the description of some product.

Value-use It is used to reserve the right to perform transactions that use a given value for some fields. For example, a salesperson may reserve the right to sell some product for a given price, even if the price is updated.

Reservations held by mobile units are leased, i.e., limited in time, thus guaranteeing that the system will be able to use the reserved values after a limited period of time, even if the mobile unit becomes permanently disconnected. However, while reservations are valid, the reserved values can not be used by any other transaction. Due to this reservation mechanism, the result of a transaction can be correctly established in the mobile unit if it only depends on reserved values (assuming that the mobile transaction can be propagated to the server before the expiration of involved reservations).

In the example presented in Figure 1, a salesman may request several reservations to be able to guarantee his decisions even while disconnected. First, he may request escrow and value-use reservations over some products to be able to guarantee orders — using both reservations he can guarantee not only that the product is available but also a given price. Second, he may request slot reservations to be able to schedule new demos with the visited clients, without the danger of having multiple demos scheduled for the same periods of time (remember that the sales department may also schedule new demos). In the next section we detail the outlined mechanism and describe the global model for transaction processing in Mobisnap.

3 System Model

The Mobisnap system manages information structured according to the relational data model. Its architecture is based on the extended client-server model [3]. The server component is composed by a mostly connected server that holds the primary copy of all data items. The clients are devices that locally replicate a subset of the database state. They are allowed to continue their normal operation even while disconnected from the server. Clients can be mobile or stationary computers. In most situations the server will be a stationary computer but nothing prevents the server from being mobile, as long as the mostly connected assumption holds.

Clients maintain two copies of the replicated data: committed and tentative. The committed version contains data received directly from the server and it reflects a possibly outdated database state. The tentative version is based on the committed one and it reflects the execution of previously submitted mobile transactions in the client unit. While disconnected, applications may access both database versions. Applications should reflect the possible weak consistency of data to users, and they should use the tentative data version to present the expected data evolution.

When clients interact with the server to fetch data copies, they can also request data reservations. In the previous section we have already defined the different types of reservations that the Mobisnap system can handle. For each specific database, the database designer should specify the associated reservation script. This script specifies the data elements that can be reserved. For each data element, it defines the type of reservations available. In escrow reservations, it also defines the number of instances that can be reserved by each client. Finally,

the reservation script specifies to whom and for how long can a reservation be granted. It should be noted that the efficiency of the reservation mechanism depends highly on the adequate definition of the above parameters for each specific system. It is also important that clients request the adequate reservations for their operation.

As usual, users manipulate the database information using applications that run on client units. These applications display data and provide operations to modify the database state through a graphical user interface. In consequence of each performed operation that modifies the database state, the application creates a mobile transaction instantiating a previously defined template with the values specified by the user. This mobile transaction program is submitted for evaluation.

If the client can communicate with the server, the mobile transaction is synchronously propagated to the server and the final result of its execution is returned to the application. If the client is disconnected, the mobile transaction is immediately executed in the client unit. The result of this execution can be one of the following:

Reservation commit This result means that the mobile transaction code has executed successfully until a commit instruction and that all tests performed during its execution are backed up by granted reservations. This result guarantees that the transaction will commit when it is finally executed in the central server if it correctly tests all dependencies and if it is propagated to the server before the expiration of the involved reservations.

Tentative commit This result means that the mobile transaction code has executed successfully until a commit instruction using the tentative database state. However, there is not enough reservations to guarantee that the transaction will commit when it is executed in the server.

Tentative abort This result means that the mobile transaction code has executed until an abort instruction using the tentative database state.

Unknown This result means that the currently cached data is not sufficient to evaluate the result of the transaction (e.g. a field or record that is referenced is not cached).

Mobile transactions are also stored by the system for later propagation to the server. By default, transactions that have been "tentatively aborted" are not propagated to the server. However, applications may request the propagation of all transactions. Transactions that have returned the result "reservation commit" are propagated to the server associated with references to the reservations used to guarantee them.

When the server receives a mobile transaction, it executes its transactional program. This execution may lead to the final commit or abort of the transaction. Besides propagating to users any messages defined in the mobile transaction, the server maintains a log recording the result of all executed transactions. Clients may access this log, if needed, to verify the result of any transaction. It should be noted that reserved values are not considered when transactions are processed

(e.g. if for some product there are 10 instances in stock and some mobile client has reserved 4 instances, only 6 instances are available for usage by other transactions). Two exceptions exist. First, the reserved values used by a "reservation committed" transaction are used and consumed when it is executed in the server — this situation guarantees that the transaction will commit. Second, when a client synchronously submits a mobile transaction for execution, this transaction may consume any reservation held by that client.

4 Related Work

In this section we will briefly overview some of the previously proposed transaction management solutions designed for mobile environments.

In Oracle Lite [7], mobile units cache database snapshots. Transactions performed in mobile units are propagated as sets of modified values (write set/old write set values) that must be integrated in the master database. Validity of transactions is checked through conflict detection — write/write, uniqueness and delete conflicts are detected. Conflict resolution functions can be associated with different database tables (or table fields). This approach has some limitation in the use of semantic information to solve conflicts — semantic information associated with updates can not be used.

In Bayou [11], data is replicated in a group of servers that synchronize their state using epidemic techniques. Bayou updates include information to allow generic automatic conflict detection and resolution through dependency checks and merge procedures. In [2], mobile nodes may propose tentative update transactions. These transactions are propagated to base nodes, where they are reapplied to the object master copy. An acceptance rule can be specified to verify the validity of transaction execution. Invalid transactions are aborted and diagnostic messages are returned to the mobile nodes. In [9], for each transaction executed in a mobile unit, the read and write sets are stored. Additionally, each transaction must specify two functions: a conflict resolution and a cost function. These functions are used to serialize transactions in the server. The conflict resolution function is always executed in the server and it can capture not only the actions of client transactions but can extend them to capture additional semantics on the server.

As in Mobisnap, the above approaches allow the use of semantic information associated with updates to solve conflicts. However, they can not guarantee the result of updates in mobile units. To solve this problem, it has been proposed the use of escrow techniques — the idea is to divide the total number of available instances of an item among different sites and/or transactions. In [5] the authors use this idea to allow mobile units to independently guarantee the results of mobile transactions. The reservation mechanism proposed in this paper includes and extends these ideas. In [12] the authors generalize the usage of escrow techniques by exploiting object semantics. However, the proposed approach can be used only with some data types and it is more adequate to object oriented databases.

5 Final Remarks

In this paper, we have presented the Mobisnap mobile transaction management model. In this model, mobile transactions performed by applications are defined in a language based in PL/SQL. These "transaction programs" are propagated and executed in the server, thus allowing the validation of transaction execution based on application-specific semantics. As mobile users are not usually connected to the system when the final result of a transaction is determined and therefore they can not immediately perform alternative actions if it aborts, these programs may contain a set of alternative actions to be executed depending on the database state. Moreover, our model provides explicit mechanisms to provide awareness information to mobile users.

One important aspect of our model is the reservation mechanism. It allows mobile units to guarantee the commitment of mobile transaction in some circumstances. We propose the definition of several types of reservations. The interested reader can obtain more information about the Mobisnap project, including an extended version of this paper that includes the system design used to implement the proposed model, from [6].

References

[1] Gray, C., Cheriton, D.: Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*, 1989.

[2] Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In *Proceedings of the ACM SIGMOD'96*, 1996.

[3] Jing, J., Helal, A., Elmagarmid, A.: Client-server computing in mobile environments. ACM *Computing Surveys*, 1999.

[4] Joseph, A., DeLespinasse, A., Tauber, J., Gifford, D., Kaashoek, M.: Rover: A Toolkit for Mobile Information Access. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 1995.

[5] Krishnakumar, N., Jain, R.: Escrow techniques for mobile sales and inventory applications. *Wireless Networks*, 3, 1997.

[6] http://asc.di.fct.unl.pt/mobisnap

[7] Oracle.: Oracle8i Lite replication Guide - release 4.0. 1999.

[8] Oracle.: PL/SQL User's guide and reference - release 8.0. June 1997.

[9] Phatak, S., Badrinath, B.: Multiversion reconciliation for mobile databases. In *Proceedings of ICDE'99*, 1999.

[10] Satyanarayanan, M.: Fundamental Challenges in Mobile Computing. In *Proceedings of the 15th ACM Symposia on Principles of Distributed Computing*, 1996.

[11] Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M., Hauser, C.: Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, 1995.

[12] Walborn, G., Chrysanthis, P.: Supporting semantics-based transaction processing in mobile database systems. In *Proceedings of the 14th Symposium on Reliable Database Systems*, 1995.