

A Unilateral Commit Protocol for Mobile and Disconnected Computing

Christophe Bobineau, Philippe Pucheral, Maha Abdallah

University of Versailles, PRiSM Laboratory
45, avenue des Etats-Unis
78035 Versailles - France
E-mail: <Firstname.Lastname>@prism.uvsq.fr

Abstract

Databases are hosted by a growing number of lightweight intelligent devices like palmtops, cellular phones, car computers and even smart cards. Mobility and disconnected computing are two major issues in such environment. This paper deals with the way databases can be exploited in this context while maintaining a strong transactional coherency. More precisely, we argue that 2PC, the most well known and well-established atomic commitment protocol, is far from being adapted to this environment. We propose a new atomic commitment protocol, called Unilateral Commit for Mobile (UCM), that exhibits the following properties : it supports off-line executions and disconnection during commitment, it decreases the cost of wireless communication by reducing the message complexity and it saves resources on lightweight servers. These properties are obtained at the price of some assumptions on the way servers manage transactions. Anyway, we show that these assumptions are not constraining in the considered environment.

Keywords: *Mobile computing, disconnected operations, distributed transactions, atomic commitment protocols.*

1. Introduction

Mobile and disconnected computing is clearly one of the most challenging area for future database systems. Personal DBMSs are commonly hosted by laptops to serve as repository for disconnected computing. More recently, databases emerge in the world of lightweight intelligent devices like palmtops, cellular phones, car computers and smart cards. Such devices can be mobile clients of fixed database servers (e.g., Web servers accessed through a cellular phone) or mobile database servers hosting personal data (e.g., phonebook, medical folder, agenda, electronic purse, ...). To tackle these issues, DBMS vendors propose lightweight versions of their products (e.g., Oracle 8i Lite [], DB2 Everywhere [IBM99]). This paper deals with the

transactional properties in this context. More precisely, it proposes a new atomic commitment protocol that eases the integration of lightweight database servers in a distributed transaction.

Traditional Atomic Commitment Protocols (e.g., 2PC) suffer from the following weaknesses when applied to mobile and disconnected computing :

- *Off-line processing* : a transaction executed on a disconnected device cannot commit locally. Thus, it remains *tentative* until the device reconnects and the 2PC is successfully executed on the fixed servers.
- *Lightweight capabilities* : lightweight servers (e.g., smart cards, cellular phones, ...) must externalize a standard interface to participate to the 2PC. This precludes the integration of legacy systems in a distributed transaction. For instance, ISO defines a transactional interface to enforce the ACID properties in a smart card but it doesn't cover distributed transactions [ISO 97]. In addition, participating to the 2PC requires building a local prepared state on each server that may consume valuable resources (e.g., smart cards).
- *Moving participants* : the 2PC incurs a significant overhead in terms of messages (2 message rounds). This can be dramatic if the coordinator is located on a mobile or if the participants themselves move. Indeed, wireless communications are much less efficient than communications on a fixed network. In addition, moving participants are likely to disconnect during the 2PC (e.g., a smart card can be extracted or a cellular phone can be put off-line or be temporarily unreachable).

To remedy to this situation, a new form of atomic commitment protocol is highly required. This paper proposes a *Unilateral Commit for Mobile* protocol (*UCM*) that exhibits the following properties :

- A transaction executing off-line can commit as soon as its log has been transferred on the fixed network and without waiting for the acknowledgment of the fixed servers.
- The protocol doesn't require the presence of all servers at commitment time.
- The protocol is composed of a single message round thereby saving costly wireless communications.
- The protocol doesn't require a prepared state nor the corresponding interface on the server side.

This paper is organized as follows. Section 2 introduces the software and hardware platform of interest and identifies the 2PC weaknesses to support different classes of mobile applications. Section 3 presents our *Unilateral Commit for Mobile* protocol (*UCM*) while Section 4 shows how this protocol tackles the specific needs of mobile and disconnecting computing. Finally, Section 5 concludes the paper and sketches future research directions.

2. Problem position

2.1 Software and hardware architecture

Figure 1 depicts a typical architecture supporting mobile computing. In this architecture, *Mobile Hosts* (MH) are connected to the fixed network via *Mobile Support Stations* (MSS). Each MSS is responsible of one radio cell, and of the handoff protocol. MH are intelligent computing devices, such as laptops, notebooks or cellular phones, which can freely move maintaining their connection to the network. The fixed network connects the MSS and some *Fixed Hosts* (FH). FH generally manage public and semi-public databases while MH manage personal databases. In terms of performance, the wireless network bandwidth is about one to two order of magnitude smaller than the fixed network bandwidth. The reader interested in technical details concerning this kind of architecture can refer to [PiSa98].

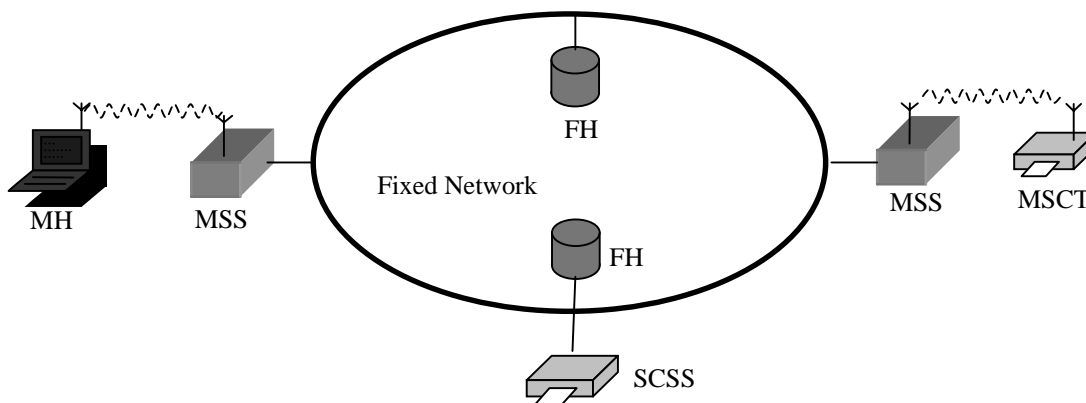


Figure 1 : Software and hardware architecture.

In addition to these traditional network components, we consider in this paper smart card based devices, due to their growing impact in many fields of the information technology. Like any other mobile hosts, smart cards own some – very light - storage and computing capabilities. However, smart cards are different from other mobile hosts because they rely on a “terminal” for communicating with the rest of the world and for their power supply. These “terminal” can be either fixed (e.g., an Automatic Teller Machine) or mobile (a cellular phone equipped with a SIM card). We call them respectively *Smart Card Support Stations* (SCSS) and *Mobile Smart Card Terminals* (MSCT).

2.2. Application’s requirements

This section presents three typical mobile computing applications, illustrating different needs in terms of transaction management.

The first application is in the field of *e-business*. Electronic shopping through Internet is

today popular and can be achieved using any fixed or mobile computer. Electronic shopping should be done in disconnected mode to decrease the communication charge (that remains quite dissuasive in Europe). To this end, the user establishes a connection with the selected supermarket and gets its catalog and quantities available for each product. The supermarket connects to the user's bank and negotiates a maximum amount authorized for his shopping. This information is forwarded to the user, which can release his connection at that time. The user now proceeds off-line (i.e., in disconnected mode), selects some products respecting the maximum amount available, and completes his transaction locally. The transaction commit is delayed until reconnection since it impacts the bank, the supermarket and even the user's computer if it keeps track of the shopping actions. Thus, the best Atomic Commitment Protocol (ACP) for off-line e-business is the one that minimizes the risk of abortion at reconnection time.

Let us now consider the use of an *electronic purse*. A regular transaction in this context consists in transferring a given amount of money from a bank account to an electronic purse hosted in a smart card. This transfer must be made atomic for obvious reasons. Here, the Atomic Commitment Protocol must address two important issues : (i) it should allow legacy devices to participate in the transaction, considering that current smart cards don't know anything about ACPs, and (ii) it should minimize the local resources required to participate in the protocol because smart card resources are tiny.

The last application consists in organizing a *rendezvous* between several persons having an agenda located on their mobile (e.g., a palmtop or a cellular phone). Let us consider that these persons are frequently - but not always - connected to the network. The rendezvous protocol is launched and the resulting selected date (assuming it exists) must be reserved atomically in all agendas. The Atomic Commitment Protocol used in this context should : (i) minimize the number of exchanged messages due to the high cost and high latency of wireless communications, and (ii) support participant's disconnection to avoid to rollback the transaction each time one of them is unreachable.

2.3 2PC background

The most widely used Atomic Commitment Protocol (ACP) is the *two-phase commit* protocol (2PC) [Gray78]. 2PC is supported by all today's commercial databases systems and TP-monitors and it has been standardized by ISO [ISO92], X/Open [X/OP91] and OMG [OMG94]. Although well known, we recall below the basics of this protocol because a precise understanding of its major steps is required in the sequel of the paper.

As depicted in Figure 2, 2PC is formed of a *Voting* phase and a *Decision* phase. During the Voting phase, the coordinator sends a Request-for-vote message (also called a Prepare message) to all the participants in the transaction. A *Yes* (or *Ready*) vote from the participant indicates that the local execution of the transaction branch was successful and that the participant is able to make its updates permanent, even in the presence of failures. In other words, the participant can locally guarantee the ACID properties of its transaction branch. This participant is said to be in a *prepared state*. A *No* vote (or *Abort*) indicates that due to some local problems (integrity

constraint violation, concurrency control problem, memory fault or storage problem), the participant cannot guarantee one or more of the ACID properties of its transaction branch.

A participant that votes No can unilaterally abort its transaction branch. If a participant votes Yes, it can neither commit nor abort unless it receives the final decision from the coordinator. During the Decision phase, the coordinator decides on the transaction and sends its decision to all the participants. The coordinator's decision is Commit if all participants have voted Yes. Otherwise, the decision is Abort (i.e., if a least one vote is No or is missing due to a process or communication failure). When a participant receives the final decision, it sends back an acknowledgment. This acknowledgment is a promise from the participant that it will never ask the coordinator about the outcome of the transaction. Once all the acknowledgments are received, the coordinator can forget about the transaction (i.e., clean its log).

The resilience of 2PC to failures is achieved by recording the progress of the protocol in the logs of the coordinator and the participants. Since failures can occur at any time, some of the information stored in the logs must be force-written (i.e., written by a blocking I/O to a stable storage that sustains failures). Indeed, the coordinator force-writes its decision before sending it to the different participants. This ensures that the decision is not lost in case of a coordinator crash. Each participant force-writes : (i) its Vote before sending it to the coordinator, and (ii) the final decision before acknowledging the coordinator. Actually, a participant that votes Yes force-writes its Vote together with all the updates performed on behalf of the transaction. This ensures that the participant is able to make its updates permanent even if it crashes (i.e., to ensure transaction Durability).

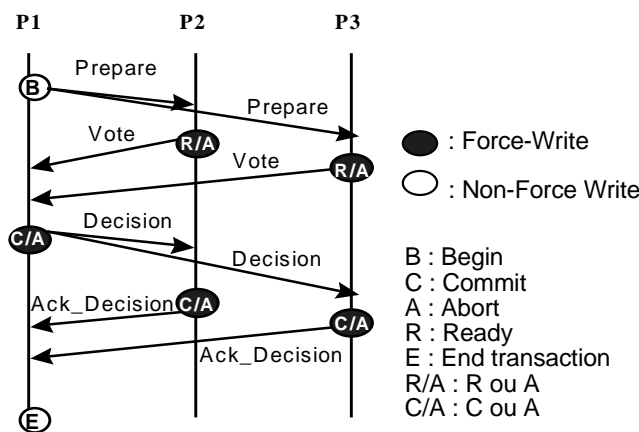


Figure 2 : The basic 2PC protocol.

2.4 2PC weaknesses

Although widely accepted and supported, the 2PC protocol suffer from strong weaknesses when applied to mobile and disconnected computing. Let's go back to the three applications introduced in Section 2.2 and see how 2PC reacts in these contexts.

If 2PC is used to commit the electronic shopping transaction, this means that 2PC is launched at reconnection time and that the transaction commit will be effective only once all

participants (the bank, the supermarket and the mobile) have acknowledged the coordinator's decision. Consequently, the mobile must retain its connection all along the protocol. Unfortunately, the transaction can be aborted even if it has been successfully executed on the mobile, due to a communication or a participant failure. The transaction can even be aborted in the absence of failure. Indeed, the voting phase of the 2PC allows any participant to unilaterally abort a transaction if it cannot locally enforce at least one of the ACID properties. This is why mobile transactions are often called *tentative* [GHOS96] until reconnection. Several answers have been given to this last problem.

The general idea is to satisfy some of the ACID properties on the mobile host (i.e., off-line). Isolation can be enforced on the mobile in different ways. Assume the mobile locks all the data extracted from the servers before disconnection, the transaction cannot be aborted at reconnection time due to a serialization problem. Obviously, strict two-phase locking (2PL) [BeHG87] cannot be used "as is" since the mobile could never reconnect. To tackle this problem, temporal constraints can be set on locks [WaCh97]. The mobile must reconnect before the deadline expires, otherwise, its locks are automatically released. Another solution consists in exploiting data semantics to partition it, so that each transaction can reserve a different partition of the same data (e.g., each transaction can reserve $x\%$ of a sharable resource). Several variations of this principle have been studied in [KrBe92], [WaCh94] and [EJB95]. Consistency can also be enforced on the mobile by checking-out the data along with the integrity constraints associated to them. This method has been promoted by the *Compact* model [WaCh97]. While these methods decrease the probability of abortion at reconnection time, a transaction can still be aborted during the voting phase because one participant cannot enforce Atomicity or Durability.

2PC is not suitable in the electronic purse application's context since it imposes each participant to externalize a *prepare* state. This means that each participant must provide a standard 2PC interface (e.g., Xa from Xopen [X/OP91]), which is not the case for current smart cards. In addition, each participant must maintain a local log to be able to conform to the coordinator's decision in any situation. This log consumes valuable storage resources in a smart card (the storage capacity of current smart cards is about 32Ko).

In the third application – the distributed rendezvous – the main weaknesses of 2PC are the following. First, 2PC doesn't sustain participant's disconnection. A disconnection during the voting phase of the protocol is treated as a crash and leads to abort the transaction (even if it could have been committed later on). Second, 2PC requires two message rounds (one per phase) and $4p$ messages, where p is the number of participants. In a mobile environment, this cost is much more constraining considering the higher cost and latency of wireless communications.

These three examples are representative of a large class of applications. They demonstrate the inadequacy of 2PC to support disconnected computing, lightweight participants and legacy servers and show that its cost is magnified in a wireless network. This makes a new Atomic Commitment Protocol dedicated to mobile and disconnected computing highly desirable.

3. The Unilateral Commit for Mobile Protocol

3.1 The case for one-phase commit protocols

The basic idea on which relies our *Unilateral Commit for Mobile (UCM)* protocol, is eliminating the voting phase of the 2PC, by enforcing some properties on the participant's behavior during the transaction execution. Thereby, the atomic commitment protocol reduces to a single phase, that is broadcasting the coordinator's decision to all participants. Roughly speaking, the coordinator acts as a dictator imposing its decision to all. If a crash precludes a participant to conform to this decision, the coordinator simply forward-recovers the corresponding transaction branch. The gain in terms of performance (blocking I/O, latency and messages) is obvious and can be largely exploited in a wireless communication network. In addition, we will show that UCM gives the ability to impose the coordinator's decision even asynchronously, thereby answering the needs introduced by disconnected computing.

The idea of One-Phase Commit (1PC) has been first suggested in [Gray78] and several variations of 1PC have been proposed more recently. The *Early Prepare* protocol [StCr90] forces each participant to enter in a *prepare* state after the execution of each operation. A participant is thus ready to commit a transaction at any time thereby making its vote implicit. The main drawback comes from the fact that each operation has to be registered in the participant's log on disk, thus introducing a blocking I/O per operation. The *Coordinator Log* protocol [StCr90] exploits the *Early Prepare* idea and avoids the blocking I/O on the participants by centralizing the participant's log on the coordinator. However, this violates site autonomy by forcing participants to externalize their log records. The *Implicit Yes-Vote* protocol [AlCh95] adapts the *Coordinator Log* to the case of gigabit-networked database systems. Finally, *NB-SPAC* [AbPu98] proposes a non-blocking version of *Coordinator Log* and preserves site autonomy by logging logical operations instead of physical records on the coordinator.

Despite its efficiency, 1PC has been largely ignored in the implementation of distributed transactional systems due to the strong assumptions it makes about the participant's behavior. These assumptions have been clearly identified and formalized in [AGP98]. Roughly speaking, 1PC eliminates the Voting phase of 2PC, which is the mean by which the coordinator verifies whether or not participants in the transaction can locally guarantee the ACID properties of their transaction branches. The governing idea of 1PC is to eliminate the need for this verification during the protocol execution by having these properties already guaranteed at commit time at every participant. This introduces the following assumptions on the way participants manage transactions :

1. 1PC protocols assume that all the transaction operations have been acknowledged (i.e., have been successfully executed till completion) before the protocol is launched. This means that the Atomicity of all the local transaction branches is already ensured at commit time.
2. 1PC protocols assume that integrity constraints are checked immediately after each update operation and before acknowledging the operation. Thus, Consistency is ensured for all the local transaction branches at commit time (no deferred integrity constraints).

3. 1PC protocols assume that all participants serialize their transactions using a pessimistic concurrency control protocol that avoids cascading aborts (i.e., strict two-phase locking [BeHG87]). This actually means that serializability (Isolation) of all the local transaction branches is already ensured at commit time.
4. 1PC protocols assume that, at commit time, the effects of all the local transaction branches are logged on stable storage, and hence the Durability property is ensured. This means that the log on the coordinator contains all the transaction updates and that this log is forced on disk before the 1PC is launched. The content of this log and the way it is exploited at recovery time differ strongly among the protocols.

We will show that these assumptions are not so constraining – and even mandatory - in mobile and disconnected computing. So, we first detail the Unilateral Commit for Mobile protocol, then we discuss how it meets the requirements of our three application samples.

3.2 Protocol description

To understand UCM, the execution phase and the termination phase of a transaction have to be considered together. Five types of components interact in the execution and termination of a transaction, namely the *Application* that asks for the execution of a sequence of operations, the *LogAgent* that logs each operation before execution, the *Participants* that execute these operations, the *Coordinator* that pilots the termination protocol and the *PAgents* (one per participant) that represent the participants in the termination protocol and play an active role during recovery. These components may be co-located or not depending on the hardware and software configuration. Note that the Coordinator is still located on the fixed network while the other components can be potentially hosted by a mobile. Figure 3 illustrates a typical configuration, where the Application, the LogAgent and the Coordinator are located on one site of the fixed network, the Participants are mobiles and their PAgents are located on Mobile Support Stations. The *rendezvous* application introduced in Section 2.2 comply to this configuration assuming the rendezvous algorithm coordinating the mobile participants executes on the fixed network.

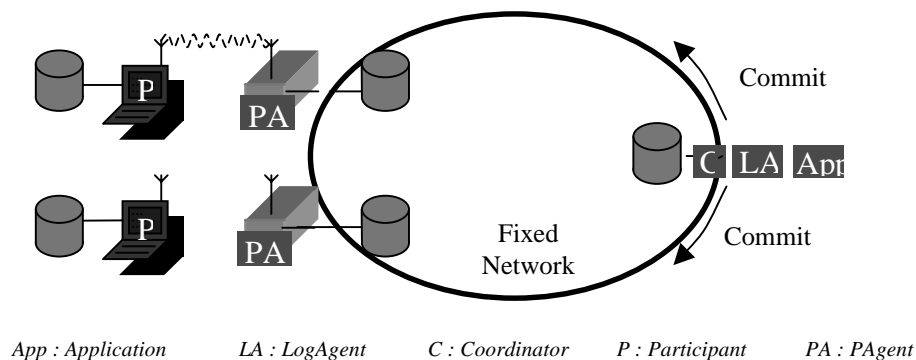


Figure 3 : A typical configuration for UCM.

The commit scenario produced by UCM is depicted in Figure 4. To simplify the figure, we show the actions executed by a single participant P_k . In the following, T_{ik} denotes the local branch of transaction T_i executed at participant P_k .

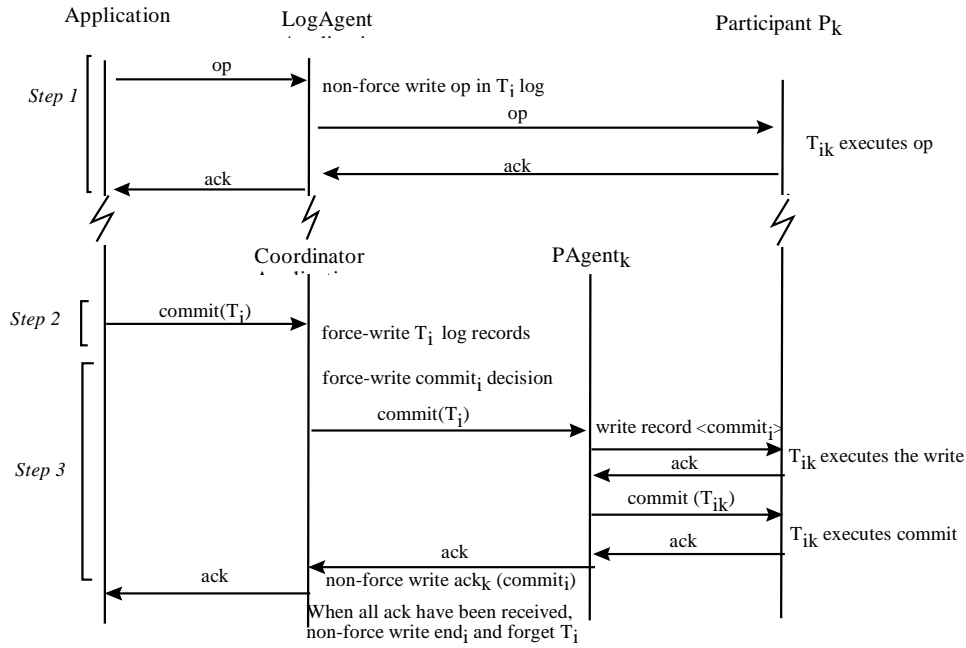


Figure 4 : Commit scenario produced by UCM.

In Figure 4, Step 1 represents the sequence of operations executed on behalf of T_{ik} . The LogAgent registers in its log each operation that is to be executed. Note that this registration is done by a non-forced write. Non-forced writes are buffered in main memory and do not generate blocking I/O. Operations are then sent to and locally executed by P_k . Every operation is acknowledged up to the application.

Step 2 corresponds to the termination of the transaction from the application point of view. Once all the acknowledgments are received by the application, it issues a commit request. At this point, properties ACI are locally guaranteed by the participants for all the local transaction branches¹. However, since participants are not aware of the termination of the transaction, they cannot guarantee the Durability property. Durability is ensured by the coordinator itself, which gets the T_i log produced by the LogAgent and force-writes it on stable storage.

Once Step 2 is achieved, the ACID properties are guaranteed altogether for all the transaction branches. Thus, the coordinator takes a *Commit* decision and force-writes this decision on stable storage (Step 3). Note that, in order to improve performances, T_i 's log records (Step 2) together with the *Commit* decision (Step 3) can be forced on stable storage at the same time, thereby generating a single blocking I/O. Then, the coordinator broadcasts the *Commit* decision to all participants and waits for their acknowledgments. When all acknowledgments are received, the coordinator forgets the transaction. The role of the *PAgents* during this step will be detailed in the next section.

¹ The participants are supposed to satisfy the assumptions stated in Section 3.1.

If transaction T_i is to be aborted, Step 2 and Step 3 are simpler than their committing counterparts. The coordinator discards all T_i 's log records and broadcasts an *Abort* decision message to all participants. We assume a presumed abort protocol, so abort messages are not acknowledged and the *Abort* decision is not recorded in the coordinator's log.

3.3. Recovery in UCM

If a participant crashes during Step 1 of UCM, the Atomicity property is violated and the coordinator simply broadcasts the *Abort* decision. Assume now that a participant P_k crashes during Step2 or Step 3 of UCM. If the coordinator has not received ack_k (commit_i), it may happen that all transaction branches have been committed except T_{ik} . Participant P_k is assumed to guarantee the local ACID properties of T_{ik} . Thus, once P_k has completed its local recovery, either T_{ik} has been backward recovered if P_k crashed before executing $\text{Commit}(T_{ik})$, or T_{ik} is locally committed if P_k crashed just before acknowledging the commit to the coordinator. These two situations must be carefully differentiated. Re-executing T_{ik} in the latter case may lead to inconsistencies if T_{ik} contains *non-idempotent* operations. Thus, if T_{ik} has been successfully committed by P_k , the coordinator does nothing. Otherwise, T_{ik} is re-executed thanks to the coordinator's log that contains all T_{ik} operations. Thanks to the assumptions made on the participants (see Section 3.2), re-executing T_{ik} (in the context of a new local transaction) is always possible and will produce exactly the same database state as the one produced by the initial execution. UCM shares the same recovery procedure as NB-SPAC, the formal proof of correctness of which can be found in [AGP98].

The role of $P\text{Agent}_k$ during the recovery procedure is to determine whether T_{ik} needs to be re-executed or not. To this end, when $P\text{Agent}_k$ receives the commit decision from the coordinator (Step 3 of UCM), it issues an additional operation "write record $\langle \text{commit}_i \rangle$ " on behalf of T_{ik} before asking P_k to commit T_{ik} . This operation will be treated by P_k in exactly the same manner as the other operations belonging to T_{ik} , that is either all committed or all aborted atomically. To get the status of a local transaction branch T_{ik} after a crash, $P\text{Agent}_k$ simply checks the existence of record $\langle \text{commit}_i \rangle$ at P_k (this can be done by a regular select operation). If the record is found, this proves that T_{ik} has been successfully committed at P_k before the crash, since $\text{write}\langle \text{commit}_i \rangle$ is performed on behalf of T_{ik} . Otherwise, T_{ik} has been backward recovered and must be re-executed.

4. Exploiting UCM in practical applications

4.1 Off-line computing

Let's come back to the *electronic shopping* application introduced in Section 2.2. The goal is to execute the shopping transaction off-line (i.e., in disconnected mode) while minimizing the risk of abortion at reconnection time. The UCM software configuration required by this application is depicted in Figure 5. The transaction executes as follows :

1. The Application runs on the disconnected mobile, co-located with the LogAgent that builds a log collecting all operations executed by the shopping transaction (Step 1 of UCM).
2. At reconnection time, the log is transferred to the Coordinator hosted by a MSS² that forces it on disk (Step 2 of UCM).
3. The Coordinator broadcasts the commit decision to all PAgents (Step 3 of UCM).

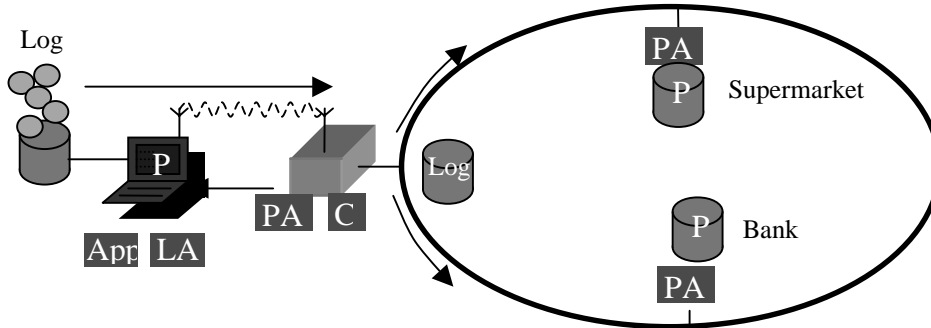


Figure 5 : UCM configuration for the e-shopping application

Note that the mobile (which acts as client and participant in this example) can disconnect from the network as soon as it has received the commit decision from the coordinator and without waiting for the acknowledgments from the other participants. The latency of the protocol – which determines the time the connection must be held by the mobile - is roughly the time needed by the Coordinator to force the log on disk. This is the lowest bound to commit a transaction executed off-line since the Durability property can be ensured only on the fixed network (a mobile can be lost or stolen).

We argue also that UCM provides the lowest probability of transaction abortion at reconnection time. Indeed, the assumptions made by UCM on the participants are those stated in Section 3.1. Thus, the ACI properties are enforced directly on the mobile during transaction execution. The transaction can be aborted only if the Coordinator crashes during the very short time where it writes its log on disk. If one participant crashes afterwards, the recovery procedure detailed in Section 3.3 is activated.

It is interesting to note that UCM and off-line computing lead to the same constraints on the participants. Indeed, the most constraining assumptions made by UCM on the participants are pessimistic concurrency control and immediate integrity control to avoid any control at transaction end. These assumptions are precisely guaranteed by techniques introduced in Section 2.4 (e.g., reservation, temporal locking and compact) to increase the probability of committing an off-line transaction.

4.2 Lightweight servers

Let's now consider the *electronic purse* application (see Section 2.2). The goal of an atomic

² An instance of Coordinator can be dynamically created on the MSS having the mobile in charge.

commitment protocol in this context is to accept legacy participants (i.e., which do not export a standard 2PC interface) and to minimize the resource consumed on lightweight participants. The best example of lightweight legacy participant is a smart card. The UCM software configuration required by this application is depicted in Figure 6. In this configuration, the electronic purse could be connected to the fixed network either via a fixed or a mobile terminal. The transaction executes as follows :

1. The Application runs on the bank site, co-located with the LogAgent and the Coordinator for security purpose. The banking operations are executed both on the e-purse side (credit part) and on the bank side (debit part) and are logged by the LogAgent (Step 1 of UCM).
2. At transaction end, the Coordinator forces the log built by the LogAgent on disk (Step 2 of UCM).
3. Then, the Coordinator broadcasts the decision to the PAgents (Step 3 of UCM).

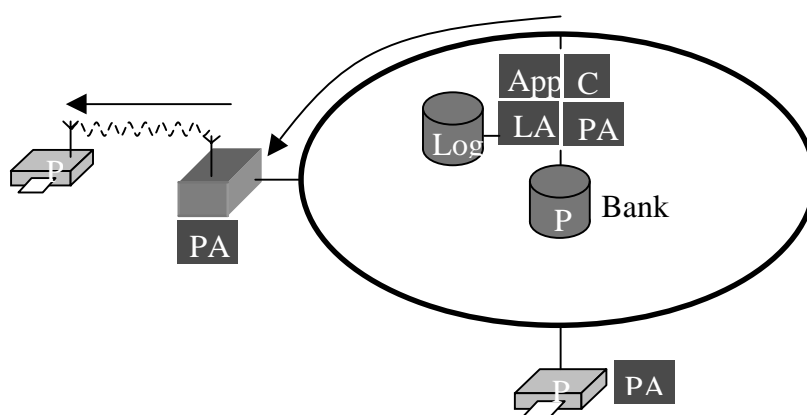


Figure 6 : UCM configuration for the e-purse application

The PAgents are in charge to translate the commit demand issued by the Coordinator into the native transactional interface provided by each participant. The bank database server is likely to support the standard 2PC interface (i.e., X/Open Xa [X/OP91]). In this case, the commit demand can be translated either into the sequence of operations *Xa_Prepate* immediately following by *Xa_Commit* or into the single operation *Xa_CommitOnePhase*. The *Xa_CommitOnePhase* operation has been designed to optimize the validation of mono-site transactions. Thus it has nothing to do with 1PC protocols but it can be diverted from its initial goal to serve this purpose. As far as we know, no smart cards providers implement the 2PC interface for obvious complexity reasons. The ISO standard for smart card [ISO97] defines a basic transactional interface enabling to begin, commit and abort a local transaction on a smart card. The mapping between the UCM commit and the smart card commit is direct. Thus, the PAgents mask the heterogeneity of the UCM participants to the UCM Coordinator and enable any kind of servers (2PC compliant or not) to participate in UCM. Note that if the smart card is extracted from the terminal during the transaction processing, the recovery protocol detailed in Section 3.3 is invoked at reconnection time to replay the e-purse operations thanks to the coordinator's log.

As said above, the mapping between the UCM commit demand and the smart card commit operation is direct. This means that none *prepare* state has to be externalized by the smart card,

thereby saving valuable resources on the card side (i.e., no log required).

One may wonder whether specific servers like smart cards satisfy the properties mandatory to participate to UCM (see Section 3.1). Under the assumption that the smart card complies with the ISO standard [ISO97] : (i) all operations sent to the card are acknowledged (via an APDU) and the card can guarantee the local atomicity of a transaction, (ii) there is no integrity constraint at all in the card (thus, no deferred control), (iii) the card uses the most pessimistic concurrency control that exists (i.e., no concurrent executions allowed). Thus, smart cards are nice candidates for UCM.

4.3 Moving topology

The last application deals with a *rendezvous* protocol launched between several mobiles. The main issues in this context are : (i) supporting participant's disconnection during the atomic commitment protocol, and (ii) minimizing the number of exchanged messages due to the high cost of wireless communications. As stated in Section 3.2., the UCM software configuration required by this application is similar to the one presented in Figure 3, assuming the rendezvous algorithm coordinating the mobile participants executes on the fixed network. The transaction proceeds as follows :

1. A mobile user launches the rendezvous protocol by proposing a date to his partners (i.e., a group of mobile users). After some iterations, all users agree on a common date and the initiator of the rendezvous asks for committing the transaction (Step 1 of UCM).
2. The Coordinator forces the log built by the LogAgent on disk (Step 2 of UCM).
3. The Coordinator broadcasts the commit decision to all PAgents (Step 3 of the protocol).

Assume one of the participants disconnects during the validation of transaction T_i and then is unable to treat immediately the commit demand from the Coordinator. Two situations have to be carefully distinguished. If the mobile continues to proceed in disconnected mode (i.e., it accepts new local transactions), the resources locally held by T_i cannot be released until reconnection. Otherwise, condition 3 stated in Section 3.1 would not be longer satisfied. Once reconnected, the participant asks the coordinator for the outcome of transaction T_i and commits it. If the participant crashes while being disconnected, it cannot recover before reconnection. At reconnection time, the recovery protocol detailed in Section 3.3 is invoked to replay T_i 's transaction branch thanks to the coordinator's log. Note that the coordinator cannot clean its log until the participant reconnects. Indeed, The coordinator is responsible for the transaction durability until it has been definitely committed on all participants (i.e., all have acknowledged the commit). In both cases, the durability of the coordinator's log allows an asynchronous validation on participants that are likely to disconnect³.

³ Note that the PAgents could be used to execute asynchronously any operations on mobile participants (i.e., they can maintain a queue of operations to be executed later on if the participant is temporarily disconnected). Anyway, this paper deals only with the atomic commitment protocol. The way operations are executed on mobiles is out of

The second issue concerns performance. Wireless communications are costly due to their very low bandwidth and to the high latency incurred by locating mobiles. The table below gives a comparison between UCM and 2PC in terms of message rounds, latency and message complexity. In this table, n denotes the number of participants and δ the time delay to deliver a message. By discarding the voting phase, UCM saves half of the messages required by the protocol and divides its latency by three. The latency of an atomic commitment protocol is the time spent between the submission of the commit demand by the application until the reception of the decision by the participants. This factor is important since it determines the time after which a participant can relax the resources held by a transaction.

Protocol	Message rounds	Latency	Message Complexity
UCM	1	δ	$2n$
2PC	2	3δ	$4n$

Table 1 : UCM versus 2PC

5. Conclusion

This paper has demonstrated through three typical examples that the well known 2PC protocol is inadequate to support mobile and disconnected computing. To meet the requirements of the growing number of applications using mobile and lightweight intelligent devices, we have proposed a new atomic commitment protocol called UCM (Unilateral Commit for Mobile). UCM exhibits the following properties:

1. It supports off-line transactions and minimizes the risk of abortion of such transaction at reconnection time.
2. It can commit a transaction despite the disconnection of one or more participants during the execution of the protocol.
3. Its message complexity is quite low (a single phase to commit the transaction), thereby saving an important communication cost in a wireless environment.
4. It applies to any kind of servers (2PC compliant or not) and do not waste the local resources of lightweight servers.

These good properties are obtained at the price of some assumptions made on the way participants manage local transactions. The most constraining assumptions are pessimistic concurrency control and immediate integrity control to avoid any control at transaction end. These assumptions are precisely guaranteed by traditional techniques used in disconnected computing. In addition, lightweight servers such as smart cards, cellular phones or palmtops are likely to satisfy these assumptions because they do not support parallelism (i.e., one user at a time) and the applications running on these devices can be considered sufficiently simple to avoid

the need of deferred integrity constraints.

For all these reasons, UCM appears to be an excellent candidate to manage distributed transactions in environments where mobility and disconnection are usual. We are planning to develop a prototype of UCM in the short term and to test its adequacy on an experimental platform developed in cooperation with Bull CP8, one of the world's leaders in smart card technology.

Acknowledgments

We wish to thank Rachid Guerraoui from Ecole Polytechnique Fédérale de Lausanne (EPFL) for its active participation in previous works on One-Phase Commit protocols.

References

- [AbPu98] M. Abdallah, P. Pucheral, "A *Single-Phase Non-Blocking Atomic Commitment Protocol*", Proc. of the 9th International Conference on Database and Expert Systems Applications (DEXA), Vienna, August 1998.
- [AGP98] M. Abdallah, R. Guerraoui, P. Pucheral, "*One-Phase Commit: Does it make sense ?*", Proc. of the International Conference on Parallel and Distributed Systems (ICPADS), Taiwan, December 1998.
- [AlCh95] Y. Al-Houmaily and P. Chrysanthis, "*Two-phase Commit in Gigabit-Networked Distributed Databases*". Proc. of the 8th International Conference on Parallel and Distributed Computing Systems, September 1995.
- [BeHG87] P.A. Bernstein, V. Hadzilacos, N. Goodman., "*Concurrency control and recovery in database systems*", book, Addison Wesley Publishers, 1987.
- [EJB95] A. Elmagarmid, J. Jing, O. Bukhres, "*An Efficient and Reliable Reservation Algorithm for Mobile transactions*", Proceedings of the 4th CIKM conference, 1995.
- [GHOS96] J. Gray, P. Helland, P. O'Neil, D. Shasha, "*The dangers of Replication and a Solution*", Proceedings of the ACM SIGMOD, pages 173-182, 1996.
- [Gray78] J. Gray, "*Notes on Database Operating Systems*", Operating Systems: An Advanced Course, LNCS, vol. 60, Springer Verlag, 1978.
- [IBM99] "*DB2 Everywhere for Windows CE and Palm OS Software, Administration and Application Programming Guide*", IBM Corp. SC26-9675-00, 1999.
- [ISO92] ISO, "*Open System Interconnection - Distributed Transaction Processing (OSI-TP) Model*", ISO IS 100261, 1992.
- [ISO97] ISO, "*Interindustry Commands for Structured Card Query Language (SCQL)*", ISO 7816-7, 1997.
- [KrBe92] N. Krishnakumar, A.J. Bernstein, "*High Throughput Escrow Algorithm for Replicated Databases*", Proceedings of the VLDB conference, 1992.
- [OMG94] Object Management Group, "*Object Transaction Service*", OMG Document 94.8.4, OMG

editor, August 1994.

- [PiSa98] E. Pitoura, G. Samaras, “*Data Management for Mobile Computing*”, book, Kluwer Academic Publishers, 1998
- [StCr90] J. Stamos and F. Cristian, “*A Low-Cost Atomic Commit Protocol*”, In Proceedings of the 9th IEEE Symposium on Reliable Distributed Systems, October 1990.
- [WaCh94] G. Walborn, P.K. Chrysanthis, “*Using the Escrow Transactional Method to Manage Replicated Data in Disconnected Mobile Operations*”, CS Technical Report 94-32, University of Pittsburgh, 1994.
- [WaCh97] G. Walborn, P.K. Chrysanthis, “*PRO-MOTION: Support for Mobile Database Access*”, Personal Technologies Journal, September 1997.
- [X/OP91] X/Open CAE Specification, “*Distributed Transaction Processing: the XA Specification*”, XO/CAE/91/300, 1991.