Distributed and Parallel Databases, 16, 193–230, 2004

A Survey of Mobile Transactions

PATRICIA SERRANO-ALVARADO* patricia.serrano-alvarado@imag.fr
CLAUDIA RONCANCIO claudia.roncancio@imag.fr
MICHEL ADIBA michel.adiba@imag.fr
LSR-IMAG Laboratory, BP 72, 38402 St. Martin d'Hères, France

Recommended by: Patrick Valduriez

Abstract. Transaction support is crucial in mobile data management. Specific characteristics of mobile environments (e.g. variable bandwidth, disconnections, limited resources on mobile hosts) make traditional transaction management techniques no longer appropriate. Several models for mobile transactions have been proposed but it is difficult to have an overview of all of them. This paper analyzes and compares several contributions to mobile transactions. The analysis distinguishes two groups of models. The first group includes proposals where transactions are completely or partially executed on mobile hosts. In this group we focus on ACID properties support. The second group considers transactions requested by mobile hosts and executed on the wired network. In this case, ACID properties are not compromised and focus is on supporting mobile host movements during transaction execution. Discussions pointing out limitations, interesting solutions and research perspectives complete this paper.

Keywords: mobile transactions, databases, mobility, transaction execution, atomicity, consistency, isolation, durability

1. Introduction

The omnipresence of mobile devices such as cell phones, PDAs (Personal Digital Assistant), smartcards, sensors and laptops together with the development of different kinds of networks (local, wireless, ad-hoc, etc.) leads to a true mutation in the use, design and development of current and future information systems. Several efforts are devoted to improve data management in mobile environments and solutions have been proposed in distinct areas [41, 58, 64]. Important topics include: broadcast to deliver information, caching, location dependent queries and mobile transactions. Research works on these topics attempt to improve support for different types of applications that can be personal or professional ones. For instance, clients accessing public data (e.g. weather forecast, stock exchange, road traffic) or with an inherent mobility (e.g. mobile vendors/clients, health care services, mobile offices, transportation).

To manage data correctly, support for traditional properties of transactions—atomicity, consistency, isolation and durability—is needed or should be revisited with respect to the specificities of these mobile applications. For that several models have been proposed so

*Supported by the CONACyT Scholarship Program of the Mexican Government.

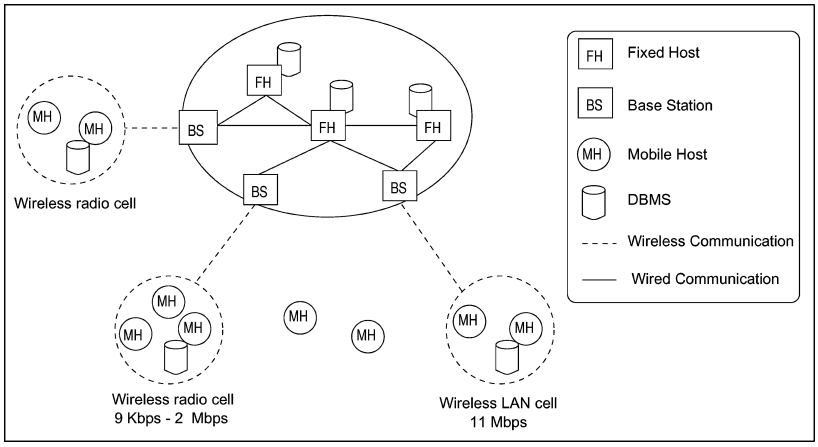

Figure 1. Mobile environment global architecture.

far [10, 16, 29, 45, 52, 61, 78–80]. These works point out the limitations of current transaction technology for mobile information systems. This paper[1] surveys mobile transaction processing works. We consider that this topic is very important and challenging, given the complex aspects of the considered environment. Despite many research efforts [16, 17, 32], the field still lacks a deep comparison of existing works. Our survey attempts to provide a global yet precise view of the current state of the art.

As a general framework, we are considering a mobile computing environment consisting of fixed (FH) and mobile hosts (MH) [38] (figure 1). Mobile and fixed hosts can be either clients or servers and MHs can be of different natures, from PDAs to personal computers. Here, we make no specific hypothesis about the database model (relational, object) and the centralized or distributed nature of the database management system (DBMS). However, we consider that MHs can have storage capabilities and can run DBMS modules. While in motion, a MH can retain its network connections through a wireless interface supported by some FHs that act as Base Stations (BS)—also called MSS. The geographical area covered by a BS is called a cell. Each MH communicates with the BS of the cell it belongs to. The process of entering a new cell is called hand-off or hand-over. Compared to traditional ones, wireless networks have particular characteristics like low and variable bandwidth. These characteristics and the (generally) expensive transmission cost make bandwidth consumption an important concern. Also, being portable devices, MHs are often limited in screen size, battery power, processing capability and memory/storage capacities, and moreover, they are exposed to loss and accidents. Communication capabilities between MHs and BSs are asymmetric: BSs do not have power constraints and can take advantage of the high-bandwidth broadcast channels that can exist from BSs to mobile clients in a cell. No direct communication among MHs is supposed. It is also important to point out that in wireless networks, bandwidth availability leads to different quality of connections—strong, weak connection. In addition, disconnections must be handled as "normal" situations and not as failures—MHs may disconnect to save battery consumption.

Most of these characteristics can impact data management process. For instance, consider an e-shopping application that allows people to browse products in an e-mall, to select, to book and to buy items. We assume that secured e-payment is available based on credit cards or *e-money*. Application execution—as a set of transactions—will not be the same if transactions are launched from a (fixed) terminal office, from a PDA while traveling in a train or from home using a laptop. Thus, under this context:

- transactions may succeed but with different execution times (bandwidth capacity is highly variable) and communication costs (prices vary among wireless network providers/ technologies/time-access);
- energy consumption is affected by low bandwidth (more battery is consumed);
- transaction failures may occur due to unexpected disconnections or battery breakdown.

Informally, a transaction is a set of operations that translate a database from a consistent state into another consistent state. In the context of mobile computing, there may exist several interpretations of mobile transactions, but here, we adopt a general definition: *a mobile transaction is a transaction where at least one mobile host is involved*. Transaction managers must be designed to support transaction processing while MH changes location and network connections. This affects mobile application performance. As we will see in the following, traditional transaction managers should be adapted to support MTs. Different transaction models (relaxing ACID properties) have been proposed to fulfill mobile user requirements and wireless environment constraints.

The rest of the paper is organized as follows. Section 2 introduces general aspects related to transaction support in mobile environments. Section 3 introduces research projects and commercial products for providing mobile transactions. Section 4 describes research works that propose special features with respect to ACID properties. Section 5 discusses research proposals dealing particularly with mobility and disconnections. Section 6 concludes the paper and gives some research perspectives.

2. Mobile transaction context

This section discusses some features related to the context of mobile transactions. Section 2.1 introduces system architecture issues. Section 2.2 deals with operation modes. Section 2.3 presents different execution models for mobile transactions. To finish, Section 2.4 briefly discusses the limitations of traditional transaction techniques in a mobile context.

2.1. Architectural context

In classic client-server architectures, functions of each actor are statically defined [40]. In the absence of failures it is assumed that neither the client and server locations nor the connection between them change. In mobile environments, however, the distinction between clients and servers can be temporarily blurred resulting in an extended client-server model [70]. Architectural choices impact application design and data management.

In particular, transaction execution on a MH is only possible if the MH provides some minimal capabilities. Data stored on MHs (memory/disk) generally come from database servers (FHs). Therefore, MH work must remain consistent with the database server. Mobile clients may vary from thin to full clients, depending on their characteristics as follows:

Thin client architecture, where clients require to run operations on servers. This architecture is specially suitable for dumb terminals or small PDA applications. Thin client resources are limited (e.g. small screen size, small cache memory, limited bandwidth). For this architecture, the server is in charge of all computations while clients only display text and graphics, play audio and compressed video, capture pen input, etc.

Full client architecture. In mobile environments, clients can be forced to work in disconnected mode or with weak connections (due to low bandwidth, high latency, or high costs). Full clients emulate server functions to enable application execution without being strongly connected to remote servers. Full clients are usually portable computers with enough resources to execute applications.

Flexible client-server architecture. Generalizes both thin and full client approaches. The roles of clients and servers as well as the application functionalities can be dynamically relocated. The distinction between clients and servers may be temporarily blurred for performance and availability purposes.

Client-agent-server architecture. This three-tier model introduces an agent or proxy located on the fixed network. Agents are used in a variety of roles acting as a surrogate of one or several mobile hosts or being attached to a specific service or application (e.g. database server access). As we will see in the following, several proposals concerning mobile transactions management adopted this architecture.

Another related, somehow orthogonal issue is *push-based data delivery*. In many wireless networks the server is connected to BSs. BSs have a relative high-bandwidth channel that supports broadcast delivery to all mobile clients located inside the geographical area they cover. This facility provides the infrastructure for push-based data delivery where the server broadcasts data to a client population without specific request. Several works are done to exploit broadcast facilities [1, 12]. In particular, push-based data delivery allows read-only transactions to be processed by a mobile client without contacting the server. Data broadcast is scalable because its performance is independent of the number of clients [62, 63]. Examples of applications are road traffic management systems, electronic commerce, business (auctions or electronic tendering, stock quotes, sport tickets). Broadcast related topics are out of the scope of this paper.

2.2. Operation modes of a mobile host

In distributed systems, hosts can operate either in connected or disconnected modes. In mobile computing new operation modes are introduced to take into consideration communication variability and MH's power limitation [59]: *strong* and *weak* connections, *disconnected*

and *doze* modes. Going from strong to weak connections depends on bandwidth availability. The degradation of connectivity is very frequent in wireless networks; thus, working with weak connections should be one of the normal operation modes in mobile systems. MHs can decide to switch to a disconnected mode to minimize connection expenses or because no connection is possible. MHs can change to a doze mode to save energy. In this mode the clock speed is reduced and no user computation is performed. The MH returns to normal operation upon the receipt of any message.

In mobile computing two types of mobility—*micro* and *macro* mobility—are generally considered, according to the scale of the movements. Micro mobility concerns movements within a geographically contiguous area. The mobile user remains in the same wireless network. For instance a connection is maintained as the user walks or moves in a car/train. Macro mobility is related to wide area mobility and can involve heterogeneous networks. The mobile user connects/disconnects his (her) laptop/PDA at different places—at work, at home or in different cities when traveling for instance.

2.3. Execution models

Mobile transactions involve MHs and FHs. As we have seen in Section 2.1, servers generally run on FHs (wired network) and MHs can be simple clients with some server capabilities. According to client capabilities we define five execution models. The first three models involve one MH whereas the fourth and fifth ones involve several MHs.

2.3.1. Complete execution on the wired network. Here, the MT is initiated by a MH but entirely executed on FHs. This is the classical query shipping approach where the data server executes update/query requests and sends results to the client. Examples in a mobile context where this execution scenario is appropriate are location dependent queries (e.g. *hotels located within a radius of 5 miles*) and updates (e.g. *booking a room in one of those hotels* [18]). In this context it is also suitable to execute transactions on a large data set.

2.3.2. Complete execution on a MH. In this case the MT is initiated and executed on the MH. This model requires the MH to have all relevant data and enough "server" capabilities to execute its local transaction. The autonomy of the MH allows it to keep working even though connections with the server are not available. Reconciliation procedures are necessary to integrate MH's work in the database server located on FHs. The following sections show that different reconciliation strategies can be adopted. In most cases, some final work has to be done on the database server even if the MT is executed on the MH. For instance, consider a salesperson having on her (his) MH all the required data related to the products she (he) sells (available stock, price, etc.). The work done autonomously (sales) on the MH is integrated afterwards to the main server.

2.3.3. Distributed execution between a MH and the wired network. This model is very flexible as it allows to distribute transaction execution between the MH and the database server(s) on the wired network. This distribution may be motivated by resource availability (e.g. data, power on the MH) or for optimization reasons. Server capabilities are required on

the MH as well as minimum communications with the server during transaction execution. The sales example introduced before may also require a distributed execution scenario. As far as the salesperson has the product information on her (his) MH, she (he) could sell products without connection to the database server—warehouse store. Nevertheless, the payment procedure may require a connection to the bank server to check client's credits. The sale is done by a distributed transaction having one sub-transaction executed on the MH and another one executed on the bank server.

2.3.4. Distributed execution among several MHs. This case is very ambitious and difficult but interesting. The objective is to provide a "peer to peer" approach. MHs act as servers for other MHs so that the execution of a MT is distributed among several MHs. The idea is that depending upon MH location, it could be interesting to ask a "neighbor" MH to act as a data server or as a service provider. Here "neighbor" means closer in terms of communication than the database server. As an example consider two salespeople working in the same geographical area who need to share some data in a cooperative way without referring to the main database server.[2] To support this execution model, particular features are required to allow MHs to be aware of each other. BSs could play an important role by maintaining specific database catalogs allowing MHs to know the data available in the area. These catalogs should be updated and forwarded across BSs according to MHs movements.

In ad-hoc networks [24, 37], MHs interact by establishing a point to point connection bypassing BSs. Since no BS is involved each MH has to maintain its own database catalog and to allow its neighbors to access it. Distributed transaction execution among several MHs is particularly oriented towards dynamic network configurations.

2.3.5. Distributed execution among MHs and FHs. This is the fully distributed scenario where MT execution is distributed among several mobile and fixed hosts. This approach is an extension of the previous scenario and is oriented towards cooperative work as well as to multidatabases including MHs participating in the global execution.

Electronic commerce is a promising application where small devices will engage in commercial transactions among themselves (execution model 4), with BSs or with remote hosts reached through a combination of wireless and wired infrastructure (execution model 5, hereafter) [66]. For instance participant MHs could be in an open air trade fair where suppliers, manufacturers, retailers and customers would meet to see the latest products available. Customers may want to buy some of the products after seeing them and consulting an online catalog available at the fair. Upon the receipt of an order the merchants could contact one another to order parts and locate supplies. With the appropriate devices and transactional support all these operations could be performed on site and be uploaded later onto the company's servers.

The five execution models introduced in this section cover all the possibilities involving mobile or fixed hosts. Nevertheless, the following sections show that current proposals concern only the first three models. Participation of several mobile hosts in a same distributed transaction has not been developed yet.

2.4. Unsuitability of classical ACID techniques

ACID properties (*Atomicity*, *Consistency*, *Isolation*, *Durability*) [31] are the reference to transaction correctness. The atomicity property is the notion of *all or nothing*—all operations of a transaction must be done or none of them. Commit protocols ensure this property. Consistency deals with database correctness. Databases usually have data integrity constraints (e.g. restrictions concerning relationships among data) for maintaining specific consistent database states. Hence each transaction must transform a database from a consistent state to another one. The isolation property ensures that each transaction execution be isolated even though transactions are executed concurrently. Concurrent control protocols ensure this property. Durability is the condition stating that once a transaction commits, its effects on the database are durable. Logging is one of the most used techniques to guarantee durability.

Previous sections introduced particular characteristics of the mobile context. This section briefly points out why these characteristics make some classic transaction techniques not suitable for mobile environments.

2.4.1. Commit protocols. Transaction commit in mobile computing is particularly affected by the unpredictable and undefined periods of MHs disconnections. If a transaction required by a MH client is completely executed on the wired network (model 1 in Section 2.3), commit is rather simple. A transaction can commit even if the MH is disconnected, and pending messages or results delivery may be deferred to the next reconnection. Consider now a mobile transaction executed on the MH using data coming from a server on the wired network (model 2 in Section 2.3). In this case, updates performed on the MH have to be integrated and committed on the server. An immediate commit on the server may not be possible because of disconnections. Deferred commits on the server will be necessary, although this may increase the transactions abort rate.

Commit protocols of distributed transactions (models 3, 4 and 5 in Section 2.3) should also be revisited. The standard Two Phase Commit protocol (2PC) [28]—used in distributed transactions—requires a high rate of messages and does not allow off-line processing. Besides, it is a blocking protocol because participants wait for a global decision. These characteristics associated to MH limited resources, disconnections and communication cost, make 2PC not suitable for mobile environments.

2.4.2. Concurrency control mechanisms. Frequent disconnections and communication limitations also affect isolation support. Pessimistic approaches such as Two Phase Locking (2PL) [22] require message exchanges with the server that may not always be possible. Unpredictable disconnections may also lead to undefined locking periods of time. Thus, optimistic approaches appear more suitable for the mobile context.

2.4.3. Logging. Fault tolerance strategies are particularly important in mobile environments. In disconnected mode, "local durability" is the base to ensure global durability on database servers. Local processing information stored in logs can be used in the reconciliation process to obtain the final commit on the database server. Nevertheless, to be applied to mobile environments, traditional logging solutions [4, 30] should be adapted. Optimizations

to reduce log size and processing time become crucial because of resource limitations at MHs. Moreover, due to the vulnerability of MHs to loss and accidents, strategies to store logs on stable storages connected to the fixed network should be proposed. Typically, periodic log transfers from MHs to current BS or FHs should be enabled.

2.4.4. Replication. Although data replication is not necessarily a transactional issue, it is at the heart of several works on mobile transactions. The reason is that common approaches to increase MHs autonomy are based on data replication or data caching. In traditional database environments coherency requirements often lead to replication protocols implementing a *one-copy equivalence* model. In this model replicas are always equivalent and read-on replicas give up-to-date data. In mobile environments, protocols that support it (i.e. eager protocols like quorum-based [43] or ROWA [3]), have to deal with the aforementioned communication/resource constraints. Such constraints motivate the use of different types of replicas (e.g. a replica may be smaller than the original object) and of protocols relaxing the coherency model (i.e. lazy protocols [29]). Such protocols may offer convergent replicas but allow reads—or even in some cases writes—on divergent replicas, in which case some kind of reconciliation strategy is required.

In conclusion, communication requirements should be reduced and protocols should tolerate disconnections. It is also worth noting that several research contributions in mobile transactions are motivated by applications where mobile users need data management but accept to work with less "guarantees" than in a wired network. In this case, applications work with advanced transaction models, relaxing some of the standard properties to provide a tradeoff between resource consumption and quality of service in a constrained environment.

3. Mobile transaction proposals

This section introduces the main projects on mobile transactions. The analyzed research projects are: Clustering (Section 3.1), Two-tier replication (Section 3.2), HiCoMo (Section 3.3), IOT (Section 3.4), Pro-motion (Section 3.5), Reporting (Section 3.6), Semantics-based (Section 3.7), Prewrite (Section 3.8), Kangaroo (Section 3.9), MDSTPM (Section 3.10), Moflex (Section 3.11) and Pre-serialization (Section 3.12). A more detailed analysis of these projects is made in Sections 4 and 5. Section 3.13 introduces briefly some commercial products. Section 3.14 summarizes the principal characteristics of studied proposals.

3.1. Clustering

Clustering [60, 61] offers a replication scheme to mobile environments where mobile clients suffer of disconnection variations. It assumes a fully distributed system and is designed to maintain database consistency. The database is dynamically divided into clusters, each one groups together semantically related or closely located data. A cluster may be distributed over several strongly connected hosts. When a 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 global

inconsistency but must be locally consistent. The degree of inconsistency is a limited divergence between the weak and the strict version.[3] Such inconsistency may vary depending on the availability of network bandwidth among clusters. MTs are either *strict* or *weak*. Weak transactions access only weak versions whereas strict ones access strict versions. Strict transactions are executed when hosts are strongly connected and weak transactions when MHs are weakly connected or disconnected. Two kinds of operations are introduced: *weak reads* and *weak writes*. Strict transactions contain standard reads and writes (strict operations), whereas weak transactions contain weak operations. At reconnection a synchronization process (executed on the database server) leads the database to a global consistent state.

Distributed transactions can be executed only inside a cluster as strict transactions. MHs may participate but only in connected mode. In disconnected mode MHs execute only weak transactions.

3.2. Two-tier replication

Two-tier replication is born from the analysis made in [29] where eager and lazy replication schemes are compared. The analysis concludes that eager schemes are not a good option for mobile environments principally because it is not possible to allow MH disconnections. Two-tier replication is a lazy replication mechanism which considers both transaction and replication approaches for mobile environments where MHs 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 access master versions (lazy-master replication scheme) whereas tentative transactions access tentative versions (local copies). Tentative transactions may perform updates on the MH in a disconnected mode. When the connection is established, tentative transactions are re-executed as base transactions (coordinated by the current BS) to reach global consistency. Results of this re-execution may have defined *acceptance criteria* which allow results to be different. Transaction re-executions allow local updates to persist.

3.3. HiCoMo

HiCoMo [49] (High Commit Mobile) is a mobile transaction model devoted to decision making applications. Its goal is to allow updates during disconnections on aggregate data stored on MHs. There exist *base tables* on FHs from which *aggregate tables* are obtained. They represent a summary or statistics (e.g. average, summation, minimum, maximum) that is stored on MHs. Similar to Clustering and Two-tier replication two transaction types are considered: HiCoMo and *base* transactions. HiCoMo transactions are executed on aggregate tables during MH disconnections. Base transactions reflect the modifications made by HiCoMo transactions on base tables. Thus, at reconnection, a HiCoMo transaction is transformed into base transactions—one per base table accessed during the generation of aggregate tables. The transformation is based on a complicated analysis using semantic information. Such a process is a key issue in this work. To obtain a high rate of successful

executions only commutative operations—addition and subtraction- are considered for HiCoMo transactions and an error margin is tolerated between HiCoMo and base transaction re-executions.

3.4. IOT

Coda [71] is a distributed file system that provides disconnected operation. It uses an optimistic replication scheme where only write/write conflicts are taken into account. IOT [50, 51] (Isolation-Only Transaction) extends Coda addressing read/write conflicts with a transaction service. In IOT, transactions are a sequence of file access operations. Transactions are classified into two categories (similar to Clustering, Two-tier replication and HiCoMo): *first class* whose execution does not contain any partitioned file accesses (i.e. the client maintains a server connection for every file accessed) and *second class* which are executed under disconnections. First class transactions commit immediately after being executed, whereas second class ones go to a pending state and wait for validation. When reconnection becomes possible, second class transactions are validated according to the desired consistency criteria i.e. local serializability, global serializability, *global certifiability* (see Section 4.3). If validation is successful, results are integrated and committed. Otherwise, transactions enter the resolution state. Resolution may be automatic (re-execution, application specific, abortion) or manual (notification to users).

3.5. Pro-motion

Pro-motion [77, 78] addresses the problem of data caching on MHs to make possible local transaction processing in a consistent mode. Pro-motion is a mobile transaction processing system that supports disconnected mode. To allow local execution, *Compacts* are introduced as the basic unit of caching and control. Object semantics is used in the construction of compacts to improve autonomy and to increase concurrency. A compact encapsulates necessary information to manage it. Pro-motion uses nested-split transactions [10, 69]. It considers the entire mobile system as one extremely large long-lived transaction executed on the server. Resources needed to create compacts are obtained by this transaction through usual database operations. Compacts construction is the responsibility of the *compact manager* at the database server. The management of compacts is performed by a *compact manager*, a *compact agent* at the MH and a *mobility manager* at the BS. The *compact manager* will act as a front-end for the database server and appears to be an ordinary database 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 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. This process checks compacts modified by locally committed transactions. If compacts preserve global consistency then a global commit is performed.

3.6. Reporting

Reporting [10] considers a mobile database environment as a specific multidatabase system (MDBS) where transactions on MHs are considered as a set of subtransactions. Nested [55] and open-nested transactions (such as sagas [26], split transactions [67] and multi-transactions [69]) are analyzed to show their limitations for mobile environments. Reporting proposes an open-nested transaction model that supports *atomic*, *non-compensatable transactions* and two additional types: *reporting* and *co-transactions* [9, 11]. While in execution, transactions can share their partial results and partially maintain the state of a mobile subtransaction (executed on the MH) on a BS. A mobile transaction is structured as a set of transactions, some of which are executed on the MH. Authors consider that limitations on MHs make necessary the use of FH, e.g. to store or compute part of a transaction. Open-nested transactions with subtransactions of the following four types are proposed. (1) *Atomic transactions* have standard abort and commit properties. (2) *Non-compensatable transactions* at commit time delegate to their parent all operations they have invoked. (3) *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. (4) *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.

3.7. Semantics-based

Semantics-based [79] focuses on the use of semantic information of objects to improve the MH autonomy 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. Mobile transactions are invoked at the MH, and from the database server point of view, they are long-lived because of communication delays. 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 server 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.

3.8. Prewrite

Prewrite [52] tries to increase data availability on MHs by proposing two variants of data: *prewrite* and *write*. A prewrite variant reflects future data state but may be structurally slightly different from the corresponding write value. Prewrite values are a tiny version of

write values; therefore they need less storage capacity on MHs. For instance, in an object of type document the prewrite is the abstract and the write is the whole document (i.e. abstract, body and bibliography). In Prewrite, the transaction execution is divided between the MH and the database server. The transaction manager on the MH executes the transaction but permanent updates are made by the data manager located at the server. Prewrite ensures that, by delegating the responsibility for write to the database server, transaction processing is reduced on the MH. Three operations—*prereads*, *prewrites* and *precommit*—to be executed by the transaction manager are proposed. Ordinary reads and permanent writes are made by the data manager. The BS has logging capabilities and maintains close relationship with the data manager. The transaction execution proceeds as follows. The transaction manager asks the BS for necessary locks (in connected mode). The BS acquires locks from the data manager and the MH can disconnect. When the transaction manager finishes the transaction, local commit (precommit) is made and reported to the BS. The data manager makes prewrites permanent and commits the mobile transaction. Prewrite considers mobile transactions as long-lived and implementations can be made with nested and split transactions.

3.9. Kangaroo transactions

KT [16] (Kangaroo Transactions) proposes a mobile transaction model that focuses on MH movement during the execution of transactions. Mobile transactions are global transactions generated at MHs and entirely executed at a MDBS on the wired network. KT proposes to implement a Data Access Agent on top of existing Global Transaction Managers. This agent is placed at all BSs and manages mobile transactions and MH movement. Each involved DBMS takes the responsibility for preserving the ACID properties of subtransactions. The transaction model uses concepts of open-nested [20] and split transactions [67]. The mobile transaction execution is coordinated by the BS to which the MH is currently assigned. When one MH hops from a cell to another (consequently from BS to BS) the coordination of the mobile transaction also moves. This mobility is captured by splitting the original transaction in two transactions (called Joeys transactions (JT), there exist one JT per BS). The split concerns only the coordination of the transaction. Thus, if the MH hops from BS-1 to BS-2, BS-1 coordinates the operations that are executed during the stay of the MH in the BS-1 cell. Subtransactions are executed sequentially therefore all subtransactions of the JT-1 transaction are executed and committed before all subtransactions of JT-2.

3.10. MDSTPM

MDSTPM [80] (Multidatabase Transaction Processing Manager architecture) proposes a framework to support transaction submissions from MHs in an heterogeneous multidatabase environment. The contribution concerning MH disconnections is the implementation of the Message and Queuing Facility that manages the exchanges between MHs and the wired multidatabase system. An MDSTPM is assumed at each host (FH) on top of existing local DBMS. Local DBMSs have the responsibility for local processing. The MDSTPM coordinates the execution of global transactions, it generates scheduling and coordinates commits.

Because of MH disconnections, a FH coordinator is designated in advance. Therefore, once a MH submits a global transaction, it may disconnect and perform other tasks without having to wait for the mobile transaction to commit. The coordinator host will manage the mobile transaction on behalf of the MH. In MDSTPM, as in KT, the manner ACID properties are enforced depends on each DBMS at each site on the wired network.

3.11. Moflex

The principal goal of Moflex [45] is to provide access to heterogeneous multidatabase systems from MHs. Moflex supports mobility management and flexibility in the definition and execution of MTs. It extends the *Flexible Transaction Model* [21] designed for heterogeneous MDBS where a transaction is a collection of subtransactions related by a set of execution dependencies: success, failure and external dependencies (time, cost or location). Besides *flexible transactions*, Moflex allows the definition of *location dependent subtransactions* [18] and the support for adaptability in the execution of subtransactions when hand-off occurs. Authors assume that the system is built on heterogeneous, autonomous MDBS. The mobile heterogeneous MDBS has three layers: MH, BS and MDBS. In the MH layer, users define Moflex transactions that are submitted to the mobile transaction manager of the current wireless cell in the BS layer. The mobile transaction manager coordinates the execution of submitted transactions. A global transaction manager at the heterogeneous MDBS layer executes transactions enforcing ACID properties.

3.12. Pre-serialization

Pre-serialization [14, 15] is oriented towards mobile autonomous MDBS. MHs request transactions from the MDBS where each DBMS runs on FHs. Mobile transactions are considered as long lived global transactions composed of compensatable subtransactions (called site transactions). Pre-serialization, unlike KT, MDSTPM and Moflex, enforces atomicity and isolation properties of global transactions while taking into account disconnections and migration of mobile users. To minimize the effects of mobile transactions (e.g. long lived executions due to mobile users disconnections) Pre-serialization allows site transactions to commit independently of the global transaction. This allows releasing local resources in a timely fashion. In addition, a pre-serialization process is executed. For this, a partial global serialization graph algorithm (PGSG) verifies the serializability of global transactions. The global transaction manager has a global coordinator and a site manager layer. The global layer consists of a set of global transaction coordinators located at each BS and at any other node supporting external users. The local layer consists of a set of site transaction managers at each participating DBMS. Global transactions are requested by mobile hosts from a global coordinator which submits site transactions to local managers. The global layer also handles disconnections, migration of mobile users, logs responses that cannot be delivered to off-line users and executes the PGSG algorithm.

3.13. Commercial products

This section introduces some commercial products proposing database solutions for mobile environments. Since they are commercial products we do not have the same technical information as we had for other (research) proposals described so far. Therefore, our discussion will stay at a descriptive level. The reviewed products are PointBase, FastObjects j2, Oracle and IBM proposals. Other proposals (not presented here) are eXtremeDB [53], Sybase iAnywhere Solutions [75] or IBM Cloudscape [35].

***3.13.1.* PointBase.** PointBase [65] is a relational database for mobile environments. Directly embedded into mobile off-line applications, it can operate on any mobile device including laptops, tablets PCs and PDAs. PointBase is entirely written in Java with a footprint (Jar file) between 45 KB and 90 KB for the *micro* version and 1MB for the *embedded* version. It can run on any platform supporting a Java Virtual Machine (JVM). PointBase supports multiple connections from only one application running on the same JVM. PointBase *micro* includes transactional support for flat transactions (auto-commit, commit and rollback operations) and partially implements the SQL92 standard and the JDBC programming interface. PointBase *embedded* supports distributed transactions (Two Phase Commit), it uses row-level locking and provides the fourth isolation levels of the ANSI SQL92 (read uncommitted, read committed, repeatable read, serializable). By using PointBase UniSync, PointBase databases can be synchronized bi-directionally with corporate databases like Oracle or Microsoft SQL Server. To store corporate data on MHs, PointBase UniSync uses a publish-subscribe model. It allows client applications to have subscriptions to published data on the server side consisting of subsets of rows and single or multiple tables.

***3.13.2.* FastObjects j2.** FastObjects j2 [23] is an object oriented database (formerly called Navajo Poet). It provides embedded database components in a 450 KB Java package. It is a single-host component for embedded applications that run in any Java compliant environment. FastObjects j2 has a modular architecture with a core that provides the essential database functionalities like object cache management and transactions. It uses object-level locking and provides the four isolation levels defined in ANSI SQL92. The modular architecture allows the incorporation of features, such as versioning, events, XML support, logging (undo-log based recovery) and replication if they are needed by the applications. FastObjects j2 allows an off-line multithreaded access to the database providing ACID, nested and parallel transactions. It supports JDOQL (Java Data Objects Query Language) and it is JDO and ODMG (Object Data Management Group) compliant. FastObjects j2 offers a "shadow database" replication feature. Replicas can be modified with transactions, changes are logged and applied to the primary database.

***3.13.3.* Oracle9iAS Wireless and Oracle9i Lite.** Oracle addresses mobile data access with the Oracle9iAS Wireless application server [57]. Oracle9iAS Wireless makes an application web site (running on FHs) accessible from mobile devices. It includes mobile services such as PIM (Personal Information Manager), e-mail, location-based (for location aware applications), and push services—via SMS (Short Message Service), WAP (Wireless Application Protocol), e-mail and voice. In particular, Oracle9iAS Wireless provides

secure mobile transactions from any mobile device (this is oriented to mobile commerce). To allow off-line processing, Oracle proposes Oracle9i Lite [56] (an extension to Oracle9iAS Wireless), a relational database with a footprint from 50 KB to 1 MB (depending on the platform). It runs under Windows CE, Windows 95/98/NT/2000, Palm OS or Symbian EPOC operating systems. Oracle9i Lite supports flat ACID transactions and the four isolation levels of the ANSI SQL92. Concurrent accesses are controlled with a row-level locking approach. Multiple JDBC or ODBC connections are supported. It is possible to create copies (snapshots) for mobile devices from Oracle master sites. Snapshots may be modified in disconnected mode. A bi-directional synchronization (Mobile Sync) with corporate databases is made (in connected mode) by the master site. Mobile Sync synchronizes multiple devices simultaneously.

3.13.4. **WebSphere Everyplace and DB2 Everyplace.** IBM has developed WebSphere Everyplace products family for data management on mobile environments. WebSphere Everyplace Access [36] allows mobile devices to access data (e-mail, PIM and business application data). It delivers web pages and e-business applications to cell phones and wireless PDAs. It contains a Location Aware Service that provides location information to mobile-enabled applications. IBM also proposes DB2 Everyplace [34], an off-line single-user relational database with a footprint of 180 KB that provides local store on mobile devices. It is available for Palm OS, Symbian OS, Windows CE, Windows 95/98/NT/2000/XP, QNX Neutrino, Linux, and embedded Linux devices. DB2 Everyplace supports a subset of SQL and provides indexing. It contains a bi-directional Synchronization Server that synchronizes relational data between the mobile device and enterprise data sources (DB2, Oracle, Informix, Sybase, MS SQL Server and Lotus Domino). DB2 Everyplace provides flat transaction management (commit, auto-commit and rollback operations). It supports multithread database connections (ODBC/JDBC) in a serialized way. In connected mode mobile devices may request queries and stored procedures executions on the database server.

3.14. Summary and discussion

Table 1 summarizes the main features of the reviewed research projects: transaction types, execution models, parts executed on MH or on the wired network and operation modes (e.g. disconnected).

All models but Reporting assume that mobile transactions are requested from MHs. In Reporting, transactions can be requested by any host. The transaction execution model (Section 2.3) used by each proposal is the following. The first model (complete execution on the wired network) is applied by Pre-serialization, KT, MDSTPM and Moflex. The second model (complete execution at a MH) is used in Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion, Semantics-based and Prewrite. The third one (distributed execution between a MH and the wired network) is supported only in connected mode by Clustering, Two-tier replication and Reporting. The fourth and fifth execution models—where execution involves several MHs—are not supported either by the research proposals or the analyzed commercial products. These products use the first or the second model and do not support distributed mobile transaction executions.

Table 1. Main features of MT research proposals.

	Transaction type	Exec. model	Execution at MH	Execution at wired network	Operation modes
Clustering	Strict and weak transactions	2 and 3	Weak transactions and *local commit* in disconnected mode. Participation in the execution of strict transactions in connected mode	Strict transactions and *commit* of weak transactions (synchronization, permanents updates)	Connected, weak connected, disconnected modes
Two-tier replication	Base and tentative transactions	2 and 3	Tentative transactions in disconnected mode. Participation in the execution of base transactions in connected mode	Base transactions	Connected, disconnected modes
HiCoMo	Base and HiCoMo transactions	2	HiCoMo transactions	Generation of aggregate tables and base transactions, execution of base transactions	Connected, disconnected modes
IOT	First and second class transactions	2	Second class transactions in disconnected mode. First transactions in connected mode	Validation and resolution of second class transactions	Connected, disconnected modes
Pro-motion	Long-lived nested-split transactions	2	The compact agent executes entirely the MT and makes *local commit*	The compact manager is in charge of compact construction, *commit* of *locally committed* transactions (synchronization, permanents updates)	Connected, disconnected modes

Reporting	Open-nested transactions with atomic, non-compensatable, reporting and co-transactions	1 and 3	Subtransactions and even global transactions	Global transactions and subtransactions	Connected mode
Semantics-based	Long-lived transactions	2	MT and *local commit*	In answer to MH requests, objects fragmentation (split) is made by the database server and also updates reintegration (merge)	Connected, disconnected modes
Prewrite	Long-lived (nested, split) transactions	2	MT and *local commit*	Lock management and *commit* of *locally committed* transactions (write operations)	Connected, disconnected modes
KT	Open-nested and split transactions	1	Transaction request	Coordination and transaction execution	Movement in connected mode
MDSTPM	Multitransactions	1	Transaction request	Coordination and execution of multitransactions	Movement in connected, disconnected mode
Moflex	Multitransactions and location dependent transactions	1	MT definition	Coordination and transaction execution	Movement in connected mode
Pre-serialization	Multitransactions	1	Transaction request	Coordination, transaction execution pre-serialization with the PGSG algorithm	Movement in connected and disconnected mode

Concerning transaction models themselves, Reporting is an original model. Authors extend the open-nested transaction model applying delegation techniques to allow visibility and to delegate MH responsibilities to FHs. In most cases contributions propose new features to support mobile transactions but do not really propose structural variations of the transaction model.

Table 2[4] shows the basic characteristics of the studied commercial products. On the one hand it is interesting to note the similarity between PointBase and FastObjects j2. Both follow the embedded approach and are 100% Java-oriented. It can be noticed that FastObjects j2 has a better transaction management mechanism whereas PointBase contains a more complete replication/synchronization process. On the other hand, mobile approaches of IBM and Oracle, besides proposing a small footprint database, address on-line web application access.

There exist also industrial products—oriented to small footprint database systems—like PicoDBMS [5] and GnatDb [76].

4. Ensuring ACID properties for mobile transactions

This section presents how MTs deal with ACID properties (*Atomicity*, *Consistency*, *Isolation*, *Durability*) [31]. Works are compared and common features identified. One section is dedicated to each property although there are no really clear borders between them. Indeed relationships between ACID properties, correctness criteria and execution protocols are not always clearly defined. The following choices were made to organize the analysis: the atomicity Section 4.1 introduces information related to commit protocols; the consistency Section 4.2 deals with the management of semantic information; the isolation Section 4.3 describes visibility and concurrency control aspects as well as mutual consistency issues (replication is discussed here). Finally, the durability Section 4.4 is mainly devoted to the durable effects of MTs and to logging aspects.

Works analyzed in this section are Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion, Prewrite, Semantics-based and Reporting. Works using execution model 1—complete execution on the wired network—propose features on mobility management and do not focus on protocols oriented to ACID properties. Thus KT, MDSTPM, Moflex and Pre-serialization are discussed in Section 5.

4.1. Atomicity

The atomicity property (Section 2.4) is ensured by commit protocols. Section 4.1.1 deals with commit process for each analyzed work. Section 4.1.2 discusses other related works such as UCM [6] and TCOT [47]. Section 4.1.3 contains a summary discussion.

4.1.1. Commit process. Except for Reporting, transaction commit is done in two steps. The first one is realized on MHs—local commit—and the second one—commit—at the BS/Database server. Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion and Prewrite execute local commit, each one with specific characteristics:

Table 2. Overview of some data management products for mobile devices.

	General information	Data access	Transactional sup.	Replication/Sync
PointBase	Relational database, 45-90 KB and 1 MB footprint, single off-line application *(execution model 2)* for any Java compliant platform	Multiple JDBC connections with support of SQL92	Flat transactions in *micro* version. Distributed transactions, four isolation levels and row-level locking in the *embedded* version	Publish/subscribe model and PoinBase UniSync with bi-directional synchronization with Oracle or Microsoft SQL
FastObjects j2	Object-oriented database, 450 KB footprint, single off-line application *(execution model 2)* for any Java compliant platform	Multiple JDBC connections with JDOQL support	ACID nested and parallel transactions (object-level locking), four isolation levels	Shadow database to backup application data
Oracle9iAS Wireless	On-line web application access *(execution model 1)*	Provides access to PIM, e-mail, location-based and push-based services		
Oracle9i Lite	Relational database with 50 KB to 1 MB footprint, off-line applications *(execution model 2)* for several operating systems	Multiple JDBC/ODBC connections	ACID transactions, four isolation levels, locking approach for concurrency control	Bi-directional snapshot synchronization
WebSphere Every-place Access	On-line web application access *(execution model 1)*	Provides access to e-mail, PIM and bussines application data, it provides a Location Aware Service		
DB2 Every-place	Relation database with 180 KB footprint, off-line applications *(execution model 2)* for several platforms	Multiple JDBC/ODBC connections with a subset of SQL. Remote queries and stored procedures execution in connected mode	Flat transactions	DB2 Everyplace Sync Server with bi-directional synchronization

- 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. 2PC). It includes participation of several hosts.
- HiCoMo, Pro-motion and Prewrite do not differentiate connected and disconnected modes. Local commit is performed using an atomic commit protocol (2PC in Pro-motion). In Pro-motion the transaction designer decides whether or not the transaction makes local commit.
- IOT also makes a local commit (in connected and disconnected modes) but recovery in case of failure is not guaranteed. Authors argue that on mobile clients, space is a limited resource and a large amount of space is needed for undoing the effect of a transaction. Consequently, the logging service will be unavailable to a client when its log space is exhausted. Locally committed transactions go into a pending state.

At the second step of the commit process, locally committed transactions make updates permanent on the database server. Transaction commit can involve reconciliation mechanisms or transaction 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, base transactions (re-execution of tentative transactions) are executed in their local commit order. If this re-execution fails, even by taking into account the *acceptance criteria* (attached to each tentative transaction), then the tentative transactions are aborted. To improve the chances of success, tentative transactions can be designed to commute with each other.
- In HiCoMo the set of base transactions generated from a HiCoMo one is organized in an extended nested transaction where each base transaction is a subtransaction. If a base transaction aborts—because of integrity constraint problems—another base transaction can be generated (from the same HiCoMo) and executed. The criteria for stop retrying depends on the defined error margin values. Thus, the global commit is almost always guaranteed thanks to the considerations made—HiCoMo transactions are commutative, error margins are tolerated and base transaction re-execution are allowed.
- IOT provides four options to reconciliate pending transactions: (1) re-executing the transaction using the up-to-date server files (this is the default option), (2) invoking the transaction's Application Specific Resolver (ASR), that is, the transaction designer may attach an ASR to a transaction to be automatically invoked by the system, (3) aborting the transaction and (4) Asking the users to manually resolve conflicts.
- In Pro-motion, compacts involved in locally committed transactions are checked. If some compacts are no more valid, then mobile transactions 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 means of the transaction processing algorithm and the locking protocol, Prewrite ensures that locally committed (precommitted) transactions will commit at the database server. This is because the prewrite and write data variants are actually different. Thus, the preread, prewrite and precommit executed on MHs are also different (but with a particular relation, see

Section 4.3) from the read, write and commit operations executed to make updates permanent.

The commit process is different in Reporting where each subtransaction is atomic but this does not imply the atomicity of the global mobile transaction. Except for *non-compensatable subtransactions*, compensating transactions can be associated to subtransactions 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 one who either commits or aborts this operation. A transaction is *quasi atomic* if all operations that it is *responsible* for are committed or none of them. Subtransactions may commit or abort unilaterally without waiting for any other subtransaction and even for their parent transaction.

In Semantics-based, transactions are considered long-lived. As MHs are responsible for local transaction commit, it would be possible to support atomic or not-atomic transactions.

Table 3 summarizes the commit processes.

4.1.2. Other commit protocols. We extend our analysis by considering other works where participants may be mobile or fixed hosts (execution models 3–5). The motivation behind these protocols generally is to provide a mobile commit process which takes into account (1) the limited characteristics of the wireless network by reducing the number of messages and (2) the mobile nature of MHs by including BSs in the commit process.

UCM (Unilateral Commit Protocol) [6] supports disconnections and off-line executions (on MHs). This work is motivated by the weakness of the 2PC protocol when executed on mobile environments: no off-line processing, MHs necessity of supporting the prepared state and 2 rounds of messages. UCM is a one-phase protocol where the voting phase of 2PC is eliminated. The coordinator acts as a "dictator" and broadcasts its decision to all participants. Several assumptions are made. For instance, all participants are required to serialize their transactions using strict 2PL[5] and at commit time the effects of all local transactions must be logged on stable storage.[6] UCM guarantees atomicity and durability; nevertheless, because of its assumptions, data accessed by uncommitted local transactions are blocked until commit and logs must be flushed (on a FH) at each transaction commit.

TCOT (Transaction Commit On Timeout) [46, 47] uses *timeouts* to provide a non-blocking protocol with restrained communication. In order to achieve this objective, instead of using messages to know if a mobile participant is ready to commit, the commit coordinator waits for timeouts to expire. The coordinator is installed on the current MH's BS, or hopping from BS to BS along with the MH. Thus, participants must send a *commit message* to the coordinator. If participant's timeouts expire and the coordinator has received neither a *commit* nor an *abort message*, the transaction is aborted. If all participant's commit are received before a global timeout, the transaction is committed without sending a global commit message. If necessary, timeouts may be renegotiated during execution. Participants may commit locally before the global commit. If the global commit fails, the coordinator sends a global abort message and compensating transactions are executed. To be able to commit independently, mobile participants have to send their logs of updates to the coordinator. When the global

Table 3. Summary of commit process.

	Commit process	
	First step at MH	Second step at BS/DB server
Clustering	Disconnected mode: *local commit* of weak transactions. Connected mode: 2PC for strict transactions	*Commit* involves syntactic reconciliation with abortion and rollback in the resolution of conflicts
Two-tier replication	Disconnected mode: *local commit* of tentative transactions. Connected mode: atomic commit protocol for base transactions	Tentative transactions are re-executed taking into account their acceptance criteria
HiCoMo	*local commit* of HiCoMo transactions	Execution of base transactions taking into account defined margin errors. If a base transaction aborts, another one can be defined and executed
IOT	*local commit* of local transactions	Four resolution options for second class transactions: re-execution, application specific, abortion and notification to users
Pro-motion	*local commit* of local transactions (2PC)	A synchronization process checks compacts involved in local transactions. In case of conflicts, local transactions are aborted and contingency procedures are executed
Prewrite	*local commit* of local transactions (prewrite operations)	Local updates are made permanent by the write operations
Semantics-based	*local commit*	Updates reintegration (merge). As fragments are exclusive copies and they have attached consistency conditions there are no conflicts in reintegration.
Reporting	All subtransactions are atomic and they are able to commit independently of the parent transaction. In case of abortion compensating transactions can be associated to subtransactions (except for non-compensatable ones)	

commit is successful, the coordinator sends MH updates to the corresponding DBMS on the wired network (reconciliation problems are not considered). The protocol considers the doze mode (see Section 2.2) but does not take disconnections into account. TCOT provides semantic atomicity [25]. To provide atomicity (not semantic atomicity) and to avoid cascading aborts, TCOT proposes using strict 2PL by each participant.

4.1.3. Discussion. Conceptually, Semantics-based, Pro-motion, Prewrite and Reporting consider transactions as long-lived ones. If these transactions are executed on MDBS, global atomicity depends on the autonomy of each database system [7]. If some DBMS cannot participate in a global atomic commit protocol, then atomicity is hard to be guaranteed. If works introduced in Section 4.1.2 are applied to MDBS, the autonomy is violated because DBMS are forced to send their logs to the commit coordinator (in TCOT) or to a fixed host (UCM). Furthermore UCM and TCOT assume that all participant processing systems use

strict 2PL (this assumption can be relaxed for TCOT). Such assumption may lead to data blocking for undefined periods of time. Nevertheless, cascading aborts are avoided.

Cascading aborts may occur in Clustering, Two-tier replication and Pro-motion. But, as local committed transactions modify local data, only aborts of local transactions are generated. In addition, these aborts concern only weak and tentative transactions because local results are exclusively available for these type of transactions. IOT prevents cascading inconsistency by notifying the users about objects accessed by non-validated transactions. In reconciliation, where a non-validated transaction is being resolved, both the local and global state for the relevant inconsistent object must be exposed, in that way the resolver can choose the local or global state for an object.

Regarding works that use a distributed execution model (Clustering with strict transactions, Two-tier replication with base transactions, UCM and TCOT), only UCM supports MH's disconnections. In Clustering and Two-tier replication MHs must be strongly connected. If a disconnection occurs the commit process raises an error. In TCOT, if a mobile participant commits locally and immediately after it disconnects, it will not be informed about a subsequent global abort.

Mobile transaction commit is generally made in two steps: *local commit* is done on MHs and *commit* is done on the BS/Database server. This approach relaxes atomicity and requires extra execution process compared to traditional techniques (e.g. 2PC). Nevertheless, it is well adapted to mobile environments because it gives MHs the possibility to work in disconnected mode without blocking the execution of the system.

4.2. Consistency

Consistency (Section 2.4) is preserved by respecting integrity constraints wich are application based—semantic information is used to define them. Section 4.2.1 analyzes the way semantic information is exploited to ensure data consistency when using mobile transactions. Section 4.2.2 summarizes the analysis.

4.2.1. Semantic information

- In Clustering, semantic information is used to specify the degree of inconsistency among clusters. This degree may be limited by (1) the number of local commits, (2) the number of transactions that can operate on inconsistent copies, (3) the number of copies that can diverge, etc. There exists also a *function h* that controls this degree by projecting strict operations on weak versions. Full consistency is achieved by merging (reconciliation) different copies of the same data located at different clusters.
- In Two-tier replication, the acceptance criteria is a test that allows base transaction results to be slightly different from tentative transactions. Such acceptable difference is semantics based. Semantic information is also used to design commutative tentative transactions.
- In HiCoMo semantic information is used: (1) to obtain aggregate tables, (2) to design commutative HiCoMo transactions, (3) to define allowed error margins and (4) to generate base transactions. In particular, the transaction transformation function (used to generate base transactions) needs as input: the aggregate table accessed, operation types, base tables

configuration, integrity constraints and conflicts between concurrent base transactions and the concerned HiCoMo one.
- In IOT, the Application Specific Resolver (ASR) solution applied to pending transactions (see Section 4.1) is based on semantic information.

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

- For Pro-motion the *compact* represents an agreement between the database server and the MH. The compact manager and the database 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 database 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. These conditions include allowable operations, constraints of their input values and conditions on the object state.

Reporting does not propose new ways to preserve consistency, but subtransactions can be related to compensating transactions—except for non-compensatable—in order to maintain semantic consistency in case of abortions.

4.2.2. Summary. Table 4 summarizes the main concepts used to preserve consistency. Semantic information on objects is essential to guarantee consistency in mobile applications. All analyzed works take advantage of 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. HiCoMo generates base transactions from commutative HiCoMo transactions, among others. IOT uses application specific resolvers to reconciliate second class transactions. Pro-motion uses semantic information to build compacts and Semantics-based to split and merge objects. Reporting bases delegation on semantic requirements, and Prewrite defines "semantically identical" data variants (prewrite/write objects).

4.3. Isolation

This section discusses three issues concerning isolation (Section 2.4): (1) the degree of *visibility* for locally committed transactions (Section 4.3.1); (2) the choice about concurrency control protocols (Section 4.3.2); and (3) mutual consistency (i.e. one-copy serializability) for data replication (Section 4.3.3). Section 4.3.4 introduces new proposals on concurrency control. Section 4.3.5 closes this part with a brief discussion.

Table 4. Summary of consistency aspects.

	Consistency and semantic information
Clustering	Definition of the function h and degrees of inconsistency
Two-tier replication	To define commutative tentative transactions and acceptance criteria
HiCoMo	To define: aggregate tables, commutative HiCoMo transactions, and error margins. To execute the transaction transformation function
IOT	ASR resolution conflicts
Pro-motion	Compacts construction (type specific methods, consistency rules and obligations) and contingency procedures
Reporting	Delegation, compensating transactions
Semantics-based	Objects fragmentation (consistency conditions and split/merge operations)
Prewrite	Definition of data variants (prewrite/write)

4.3.1. Visibility aspects. Concerning visibility of intermediate transaction results, Clustering, Two-tier replication, HiCoMo, IOT, Pro-motion and Semantics-based give visibility of locally committed results to local transactions on the same MH. Prewrite makes public locally committed results when the local commit is reported to a BS. In Reporting, visibility is allowed 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 database become visible to other transactions. The objective of reporting and co-transactions is precisely to allow visibility of partial results while in execution.

Taking Pro-motion and Reporting as open-nested transactions, global isolation is not enforced 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 database server. Results of this split (committed) transaction are visible to the whole database environment.

4.3.2. Concurrency control schemes. To manage concurrent executions, Clustering and Prewrite use 2PL-oriented protocols and propose new conflict tables.

- Clustering uses strict 2PL 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 incompatible with any

other lock. Weak transactions release their locks at local commit and strict transactions at commit. If weak and strict transactions are executed concurrently in a cluster, a correct schedule ensures that a weak read operation reads data modified by the last write (weak or strict) operation, and a strict read operation reads data modified by the last strict write.

- As Clustering, Prewrite uses a 2PL protocol and the conflict operation table includes preread and prewrite operations (PR, PW, R, W). Prewrite and preread locks concern the prewrite version of data. Read and write locks concern the write version. All locks are managed by a BS. To make prewrites permanent the prewrite lock must be converted into a write lock; in this way, the data manager can write and commit mobile transactions. Preread locks are released at local commit time whereas prewrite/write/read locks at commit time. At using simple objects (without two variants) prewrites are identical to writes and the algorithm behaves as using relaxed 2PL. Prewrite ensures that the transaction processing algorithm along with a lock-based protocol produce only serializable histories. This serializability is based on the local commit order of mobile transactions.

In HiCoMo, as HiCoMo transactions are commutative, their execution can be made without strong restrictions on the order. Whereas, base transactions are not commutative—they can execute division or multiplication operations—thus, the order between them and HiCoMo transactions is important. An optimistic concurrency control strategy based on timestamps [58] is used to detect conflicts.

Similarly, in Two-tier replication if tentative transactions are commutative there is no need of a local concurrency control mechanism. Nevertheless, for executing base transactions locking mechanisms are used.

In IOT, concurrency control is made in two levels. Across clients, global concurrency control is maintained using the *optimistic concurrency control* (OCC) schema [48]. Within a client, local concurrency control is enforced with strict 2PL with a periodic deadlock detection. Serializability is guaranteed locally.

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. Levels are characterized based upon the degrees of isolation defined in the ANSI SQL standard as extended in [2]. Level 9 represents a serial execution of transactions and level 8 a serializable execution. Each successive level represents a weaker degree of isolation. At level 0 there is no guarantee about isolation. Since the arbitrary use of isolation levels can lead to inconsistencies, Pro-motion proposes simple rules:

1. Transactions impose a minimal level for write and read operations.
2. Each operation is associated with a level.
3. None of the write operation levels is lower than the write level of the transaction.
4. None of the read operation levels is lower than the read level of the transaction.
5. The lowest level of any read operation is greater than or equal to the highest level required by any write operation.

In Semantics-based, to ensure serializability, local transactions have access to cached fragments by using conventional concurrency control protocols (e.g. 2PL).

4.3.3. Replication issues. Replication issues are tightly integrated with mobile transaction management in several works. Clustering and Two-tier replication maintain two versions of data. 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. Consistency in strict/master versions is preserved using one-copy serializability methods. To provide coherency Clustering uses quorum consensus and Two-tier replication uses a lazy-master replication protocol. In Two-tier replication, tentative data versions are discarded at reconnection because they are outdated by completely refreshed master versions.

HiCoMo, Pro-motion and Prewrite take a different approach. They construct a special type of data (from sources stored on FHs) that will be stored on MHs and that can be considered as a special kind of replica. In HiCoMo, aggregate tables are generated from base tables. The correctness criterion is *convergence*[7] where base tables eventually reflect updates made in aggregate tables. The approach is similar in Prewrite, where the prewrite variant is a smaller variant of the write value. In Pro-motion, unlike HiCoMo and Prewrite, compacts contain not only specific data but also special information for using it. The flexibility offered by compacts allows Pro-motion to support several dynamic replication schemes with a variety of consistency constraints and correctness criteria.

In IOT, a variant of the read-one, write-all approach (ROWA) [3] is used to maintain consistency in a fully connected environment. With ROWA, first class transactions are serializable with all committed transactions. In disconnected mode, an optimistic evolution on MHs is assumed. IOT defines a correctness criterion called *global certifiability*. It requires that a pending transaction to be globally serializable *with* and *after* all previously committed or resolved transactions. Global certifiability is enforced with a systematic re-execution of pending second class transactions. This is the default consistency criterion.

4.3.4. Other concurrency control approaches. As discussed in Section 2.4, 2PL is not suitable for distributed transaction executions including MHs (execution models 3–5). This is because of unknown locking time due to unpredictable disconnections. Variants have been proposed as in [54] where a concurrency control scheme integrating optimistic and pessimistic approaches is presented. A timeout is associated to locked data. This timeout is the estimated time interval within which the transaction is expected to commit. If the commit does not occur within that period (because of disconnections), then the pessimistic policy is switched to an optimistic one. In reconnection, optimistically committed transactions are re-executed.

O2PL-MT (O2PL for Mobile Transactions) [39] extends the optimistic two phase locking algorithm (O2PL) [8] to mobile environments. In this algorithm, read locks are granted on demand and write locks are deferred until commit time. Working in a replicated context, the O2PL-MT algorithm reduces the number of messages to be sent when releasing read locks. O2PL-MT allows releasing a read lock for an item to be executed at any copy site, regardless of whether that site is different from the copy site where the lock is set.

4.3.5. Discussion. Mobile computing usually involves some kind of replication because data cached on MHs are extracted from databases located on the wired network. Under data

replication two correctness levels are considered: locally (on each host) and globally (on all hosts). In a fully connected environment it is possible to ensure global correctness (or mutual consistency) of the mobile system. For instance, see Clustering, Two-tier replication and IOT in Table 5. In disconnected or weak connected modes, the global correctness criteria must be relaxed to avoid blocking MH as well as FH executions. Thus, local data evolution may continue giving MH some autonomy. Eventual global correctness [68, 69] seems to be adequate to mobile environments because consistency is achieved or required "at specific real-time," "within some time" or "after a certain data value threshold is reached". Among analyzed models note that in Clustering, eventual consistency is proposed to define degrees of inconsistency (see Section 4.2.1). This allows "a maximum number of transactions that may operate on disconnected mode," "a range of acceptable values a data may take," "a maximum number of divergent copies per data," "a maximum number of updates per data not reflected at all copies".

As we can notice in Table 5, although local results are tentative (in reconnection a global commit should be done) the majority of analyzed works makes them locally visible. It is also interesting to notice that in general local correctness is ensured by using traditional locking approaches (i.e. 2PL). Nevertheless, as 2PL is not well suited for distributed mobile executions, optimistic-oriented protocols are being developed (e.g. see Section 4.3.4).

4.4. Durability

To make mobile transaction effects durable (Section 2.4), local commits must be transformed into global ones on the database server. Section 4.4.1 shows the opportunity for local committed transactions to successfully commit at the database server level. As the majority of studied works do not address particular logging techniques, Section 4.4.2 completes this analysis with other related proposals. Section 4.4.3 gives a brief discussion of analyzed aspects.

4.4.1. Durability guarantees. Clustering, Two-tier replication, IOT and Pro-motion cannot guarantee durability before commit on the wired network. Pro-motion with compacts can give some guarantees of durability, but there may exist conditions that cannot 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.

HiCoMo, Semantics-based and Prewrite approaches guarantee durability upon local commit. Nevertheless, the first work requires commutative HiCoMo transactions, besides it has a complicated base transaction generation (see Section 4.2) and considers regenerations and base transaction re-executions (in case of abortions). Semantics-based reduces fragments availability because a MH can hold fragments for an undefined period of time. In the Prewrite algorithm, if a mobile transaction makes a local commit, its commit is assured. The drawback is the message exchanges necessary to get locks from the BSs.

Table 5. Summary of isolation aspects.

	Visibility	Concurrency control	Replication issues
Clustering	*local committed* transaction results are visible to local weak transactions on the same MH	2PL, 4 conflict tables and new lock types are proposed	2 versions of data: strict (one-copy serializability using quorum consensus protocol), weak (degrees of inconsistency, data evolution in disconnected mode)
Two-tier replication	*local committed* transaction results are visible to local tentative transactions on the same MH	Locking mechanisms for base transactions	2 versions of data: master (one-copy serializability using a lazy master replication protocol), tentative (local data evolution in disconnected mode)
HiCoMo	*local committed* HiCoMo results are visible to another HiCoMo on the same MH	Optimistic timestamp ordering for a HiCoMo with base transactions	Aggregate and base tables. Convergence as correctness criteria
IOT	*local committed* transaction results are visible on the same MH	OCC globally, 2PL locally	A variation of ROWA for a fully connected environnement, optimistic evolution in disconnected mode, global certifiability as consistency criterion
Pro-motion	*local committed* transaction results are visible to local transactions on the same MH	Possibility of different isolation levels and concurrency control per compact	Compacts definiton allows several replicaton schemas
Reporting	with subtransactions *atomic, reporting and co-transactions* visibility is allowed before the commit of the global transaction		
Semantics-based	*local committed* transactions results are visible to local transactions on the same MH	2PL to control access to locally cached fragments	
Prewrite	*local committed* transactions results are visible to all hosts	2PL extended, one conflict table and new lock types are proposed	Write and prewrite data variations

Concerning logging, IOT (actually Coda [44]) proposes a mechanism to reduce the log size. In disconnected mode, sufficient information to replay updates (when reconnecting) is maintained in a *replay log*. To reduce the length of the replay log, in update operations, instead of logging the "open," "close" and intervening "write" operations individually, a single record is logged during the "close" operation. Authors also discard previous *store* records for a file when a new one is appended. This is possible because a *store* operation makes all previous versions of a file useless.

4.4.2. Related works on logging. The Little Work project [33], like IOT, proposes to reduce the log size. It suggests to apply rule-based techniques used in *compiler peephole optimizers*. Rules are used to eliminate redundant or useless operations from logs. The optimizer takes an input list of rules, each of which consist of a set of *source* operations followed by a set of *target* operations equivalent to the source set. For instance, a "create file_i" operation followed by a "rename file_i by file_j" will be replaced by a "create file_j" operation.

The effect of mobility on logging is analyzed in [17]. The problem addressed is: if mobile transactions are distributed on several MHs, where should reside logs in order to guarantee durability? Three techniques to address this problem are proposed.

1. The *Home BS logging* approach maintains the MH logging at its home BS (the BS where the MH initially was registered) even though the MH moves and changes from BS to BS. The log of a distributed mobile transaction will be scattered over the home BSs of the participating MHs.
2. The *Home MH logging* approach stores the log at the BS covering the MH when the transaction is originated. The log of a transaction is centralized in one BS.
3. The *Local BS logging* approach stores the log entries on the current MH's BS. The entire log of a transaction may be scattered into a number of BS.

4.4.3. Discussion. Table 6 shows when durability is ensured and what drawbacks here are. Notice that generally, durability is guaranteed after commit, i.e. when local commits are reintegrated to database servers. In the case where local commit guarantees durability, drawbacks concern data availability, communication costs or complicated data reintegration.

In other respects, optimized logging techniques are not proposed. Only IOT and the Little Work project address this issue, although—as was said in Section 2.4—methods to reduce MH's log sizes are needed and more research efforts on this subject are necessary.

Because MHs are not considered as stable storage, it should be considered to allocate logs on FHs (e.g. BSs). Regarding MH's log location approaches proposed in [17] the benefits of one technique or another depend on the mobility profile of MHs as well as on the distribution of the mobile transaction execution. Section 5 addresses more aspects of this issue.

Table 6. Summary of durability aspects.

	Durability guarantees	Drawbacks
Clustering	After *commit* (reconciliation)	*Locally committed* transactions can be rolled back due to reconciliation conflicts
Two-tier replication	After *commit* (re-execution)	*Locally committed* transactions can be rolled back due to conflicts during re-execution
HiCoMo	After *local commit*	Commutative HiCoMo transactions, complicated base transaction generation, regeneration and re-executions
IOT	After *commit*	*Locally committed* transactions can be aborted due to reconciliation conflicts
Pro-motion	After *commit* (reconciliation)	*Locally committed* transactions can be rolled back due to reconciliation conflicts
Reporting	If the parent transaction *commits*, subtransactions are durable	
Semantics-based	After *local commit*	Reduction of fragments availability at database server
Prewrite	After *local commit*	Many message exchanges between MHs and BSs

5. Movement and disconnection management

The previously analyzed works do not give details about management of MH mobility. Only Pro-motion includes in its architecture a *mobility manager* that is in charge of communication between the MH and the database server without giving details about the way it works. Therefore, here is proposed a complementary analysis for Section 4.

In KT, MDSTPM and Moflex, ACID properties are not affected by mobility because transaction execution is under the responsibility of DBMSs located at FHs. However, as transactions are requested from MHs, mobility and disconnections are managed. Pre-serialization is a special case in this section because even though mobile transactions are executed on the wired network, the atomicity and isolation properties are enforced by allowing disconnections of mobile users during transaction execution.

Section 5.1 highlights the way analyzed works deal with movement and disconnections. Section 5.2 discusses and compares analyzed approaches.

5.1. Movement and disconnection issues

In **KT** to support MH mobility and disconnections, the Data Access Agent keeps track of MH movement by maintaining a list of all the BSs that have been coordinators for the

mobile transaction. This list is used in case of cascading aborts. There are also data structures (*transaction status table* and *local log*) that store information about mobile transactions such as: global transaction ID, status (active, commit, abort), Joey transaction ID, subtransactions that are included in the Joey transaction (JT), compensating transactions (if any), etc.

In KT two different processing modes are supported, *Compensating* and *Split* modes. Under Compensating mode, the failure of any JT causes the current and any preceding or following JTs to be undone. Previously committed JTs must be compensated. This operating mode requires that the user provides compensating transactions and that the source system guarantees their successful commit. In the Split mode, when a JT fails no new global or local transactions are requested. The commit and failure of currently executing transactions is a decision left to local DBMS. Split is the default mode. Neither Compensating nor Split guarantees serializability of the Kangaroo transactions. Compensating mode ensures atomicity, but isolation may be violated because concurrency control is managed autonomously at local transaction level. Notice that user's movement does not affects these properties.

In **MDSTPM** the main idea is a Message and Queuing Facility (MQF) which is an asynchronous message exchange, where messages are of types: Request, Acknowledgment, and Information. With MQF the MH can submit global transactions and switch to disconnected mode. MHs and coordinator hosts maintain tables and logs that record the overall state of the MH as well as information on global transactions (*Message Queue*, *Transactions Queue*, *Global Log*, *Global Transaction Table*, *Site Status Table*). At any moment, a MH can request information about its global transactions.

Concerning correctness aspects, in MDSTPM participating DBMS are autonomous and may be heterogeneous. Thus, local transaction management mechanisms (e.g. concurrency control) can be different and the information regarding local executions (e.g. logs) is limited or denied. To manage global transactions, MDSTPM implements strict 2PL and uses the optimistic ticket method (OTM) [27] to solve indirect conflicts [7]. In OTM all global subtransactions are forced to obtain a ticket. This causes additional conflicts between subtransactions and the execution order is determined by the ticket.

In **Moflex** two characteristics concerning mobility are highlighted: the execution of location dependent transactions [18] and hand-off influence on transaction execution. In the transaction definition, users can specify whether a subtransaction is location dependent or not. For location dependent subtransactions, hand-off control rules have to be specified. Choices are:

- *continue* the transaction execution at the new cell;
- *restart*: abort the transaction at the previous cell and restart it at the new cell;
- *split-resume*: operations executed at the old cell commit and remaining operations are executed at the new cell;
- *split-restart*: operations executed at the old cell commit and the transaction is executed entirely at the new cell.

The split operation used here is similar to the one used in KT. When MHs hop from BS to BS, transactions are split and the coordination is relocated at the new BS. Transaction

definitions may also include goal states indicating acceptable final states. The 2PC protocol is used. The mobile transaction manager of the cell where one of the subtransactions reached an acceptable goal state, becomes the coordinator of the 2PC protocol for the global commit.

In **Pre-serialization** the principle is to enforce Atomicity and Isolation (A/I) properties supporting disconnection and migration of mobile users during the execution of mobile transactions (actually global transactions). Site transactions of global ones are organized in *vital* and *non-vital* ones. A/I properties are enforced only on the set of vital site transactions. The abort of a non-vital site transaction does not force the global transaction to be aborted. The time between the submission of the first vital site transaction and the completion of the last one is called the vital phase of a global transaction. Global transactions may be in one of the following states:

- *active*, the user is connected and execution continues,
- *disconnected*, the user is disconnected but the disconnection was predicted and reconnection is expected; the execution continues,
- *suspended*, the user is disconnected and is deemed to have encountered a severe failure,
- *committed* or *aborted*, the transaction is committed/aborted.

The transaction execution is not stopped when user disconnects (in a predicted way). All responses addressed to a disconnected user are delivered upon reconnection.

The control of global transactions migrates from a global coordinator to another one according to the user's movement. The site transaction managers supervise the execution of their site-transactions. Site transactions (vital or non-vital ones) may be in one of the four states: *active*; *completed* (the site transaction has committed at the local database but the global transaction has not committed); *committed* (the site transaction and the respective global transaction have committed) and *aborted*. If the global transaction is in a disconnected state, the execution of site-transactions continues. A global coordinator stores messages for disconnected users, delivers them upon reconnection and reactivates disconnected transactions.

The global coordinator verifies A/I properties by executing the Partial Global Serialization Graph (PGSG) algorithm at the end of the vital phase. If A/I are violated, the global transaction is aborted; otherwise it is *toggled*. After being toggled, a mobile global transaction may initiate only non-vital site transactions. Once toggled, the global transaction establishes its serialization order in the global serialization schema and it is guaranteed to commit. At the end of its execution, each toggled transaction runs a second time the PGSG algorithm. The objective is to verify that non-vital site transactions do not violate the serialization order established when the global transaction was toggled. Any non vital site transaction that violates this order is aborted without affecting the global transaction. A toggled global transaction is aborted only if it prevents the execution of another global transaction while it is in the *suspended* state. Thus, Pre-serialization guarantees semantic atomicity and global serializability. A modified version of Pre-serialization is proposed, where, A/I properties are enforced only for vital transactions. In this version, the PGSG algorithm is executed once, at the end of the vital phase.

5.2. Discussion

KT and MDSTPM are very similar. They propose to add a layer in existing multidatabase architectures to manage transactions requested by MHs. The main difference is the choice of the coordinator host. In MDSTPM the coordination of the mobile transaction execution is centralized. The FH coordinator is fixed in advance and does not change during the whole execution. In KT, the coordination is distributed among all the BSs visited by the MH. Hence, we notice that KT deals with the mobile nature of MHs, not only with disconnections. Distributed coordination reduces communication cost during execution; however, in case of cascading aborts communication cost highly increases. In contrast, with a centralized coordination as in MDSTPM, cascading aborts will be easier and cheaper; however, in case of high mobility, communication will be expensive.

A good analysis about the impact of mobility on transactions requested from MHs and executed on DBMS in the wired network is provided in [19]. Three possible approaches for transaction coordination are analyzed: (1) fixed at the MH, (2) fixed at a centralized FH, and (3) moving from BS to BS. In other respects, [17] proposes a mobile transaction definition dedicated to *location dependent data*. Authors analyze the impact of mobility on such data and their effect on the ACID properties.

In mobile computing, adaptability to environment variations is an important issue. Besides hand-off adaptability, among analyzed works, only Moflex is interested in location dependent transactions. Moflex main drawback is that users have to provide a complicated transaction definition. Issues concerning adaptability are treated in [72, 73].

The singularity of Pre-serialization is that A/I properties are addressed in the multidatabase context taking into account mobile hosts disconnections. Nevertheless, the PGSG algorithm is costly because of the propagation principle (dissemination of serializability information) and also because it might be executed twice during global transaction processing (in the basic version).

6. Conclusions and research directions

This paper explored works on mobile transactions. This topic is particularly important today when information systems may involve mobile devices and fixed hosts reached through a combination of wireless and wired networks. In the last years, many academic and industrial efforts have been devoted to improve data management in mobile environments. Unlike traditional centralized or distributed environments, mobile environments are highly versatile and face several resource constraints. By consequence mobile transaction executions are not predictable and require adapted approaches. These are some of the reasons that lead to the development of various new approaches for mobile transactions.

Several research projects and commercial products were overviewed. A deep analysis of the research projects led to a classification in two groups. The first one relaxes ACID properties to allow transaction executions on MHs. Works in this group usually ignore MH's movements. The second group of proposals concerns transactions requested by MHs and executed on FHs. In this case, MH movements are taken into account during transaction execution. These approaches are discussed and summarized in several sections and tables.

This study showed that ACID properties are hard to enforce in mobile environments. The main reasons are the autonomy needed to work in disconnected or weak connected modes and the inherent mobility of MHs. Almost all analyzed projects release ACID properties in order to increase flexibility and to manage mobile environment constraints. Advanced transaction models are clearly appropriate for mobile environments.

In spite of the large number of works on mobile transactions, some research issues remain open, among them, logging and adaptable mobile transaction management.

Logging has to be revisited to face MH limited storage capacities. Log compression and access optimization are nearly ignored by the analyzed works. Also, MHs may be lost and are exposed to accidents. Strategies transferring local logs to FH's (BS or servers are considered as stable storage) are needed to guarantee durability of locally committed transactions.

Another important point is adaptability due to mobile environment variations (location, bandwidth, neighborhood, server distance) or to limitations of mobile computing resources (storage/power capacity). Almost all studied systems adapt their behavior to support connected and disconnected modes. Only a few of them consider adaptability related to MH movements (i.e. hand-off). However, variations are not limited to disconnections and hand-off. A dynamic choice of transaction execution models, depending on current location, network or MH resources, would certainly increase transaction success rate.

To conclude, it is worth noting that supports for distributed transactions involving several mobile hosts have not yet been developed. It would require a flexible management that allows a dynamic configuration of the set of participants.

Acknowledgments

The authors gratefully acknowledge the many constructive comments made by the anonymous reviewers and the editor P. Valduriez; they helped to improve an earlier version of this paper. We also thank A. Moran, N. Adiba and B. Raffin for their help in reading this paper. We wish to thank the members of the NODS project (http://www-lsr.imag.fr/Les.Groupes/STORM/) for their feedbacks all along this research.

Notes

1. This work is an extended version of [74].
2. This execution model is considered by Bayou [13] and Deno [42]. Those projects are not developed here because they are replication-oriented rather than transaction-based.
3. Section 4.3 gives more details.
4. In the following tables an empty cell means that we do not have enough information about this point.
5. Strict 2PL (the most implemented version of 2PL) releases write locks after the transaction commits or aborts, read locks can be released when the transaction terminates. Generated schedules are *strict* [4] (i.e. they are recoverable and avoid cascading aborts).
6. Physical storage of MH is not considered stable because MHs are subject to lost, damages or undefined disconnections. Thus, only FHs are considered as stable storage.
7. Convergence states that eventually all replicas will converge on the same state.

References

1. S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Prefetching from a broadcast disk," in Int. Conf. on Data Engineering (ICDE), New Orleans, USA, Feb. 1996.
2. H. Berenson, P. Bernstein, J. Gray, J. Melton, E.J. O'Neil, and P.E. O'Neil, "A critique of ANSI SQL isolation levels," in ACM SIGMOD Conference, San Jose, USA, May 1995.
3. P.A. Bernstein and N. Goodman, "An algorithm for concurrency control and recovery in replicated distributed databases," ACM Transactions on Database Systems (TODS), vol. 9, no. 4, 1984.
4. P.A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley Publisher, 1987.
5. C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez, "PicoDBMS: Scaling down database techniques for the smartcard," in Int. Conf. on Very Large Databases (VLDB), Cairo, Egipt, Sept. 2002.
6. C. Bobineau, P. Pucheral, and M. Abdallah, "A unilateral commit protocol for mobile and disconnected computing," in Int. Conf. Parallel and Distributed Computing Systems (PDCS), Las Vegas, USA, Aug. 2000.
7. Y. Breitbart, H. Garcia-Molina, and A. Silberschatz, "Overview of multidatabase transaction management," Very Large Databases (VLDB) Journal, vol. 1, no. 2, 1992.
8. M.J. Carey and M. Livny, "Conflict detection tradeoffs for replicate data," ACM Transactions on Database Systems (TODS), vol. 16, no. 4, 1991.
9. P.K. Chrysanthis, ACTA, "A framework for modeling and reasoning about extended transactions," PhD thesis, University of Massachusetts, Amherst, USA, 1991.
10. P.K. Chrysanthis, "Transaction processing in a mobile computing environment," in IEEE Workshop on Advances in Parallel and Distributed Systems (APADS), Princeton, USA, Oct. 1993.
11. P.K. Chrysanthis and K. Ramamritham, "Synthesis of extended transaction models using ACTA," ACM Transactions on Database Systems (TODS), vol. 19, no. 3, 1994.
12. A. Datta, D.E. VanderMeer, A. Celik, and V. Kumar, "Broadcast protocols to support efficient retrieval from databases by mobile users," ACM Transactions on Database Systems (TODS), vol. 24, no. 1, 1999.
13. A.J. Demers, K. Petersen, M.J. Spreitzer, D.B. Terry, M.M. Theimer, and B.B. Welch, "The Bayou architecture: Support for data sharing among mobile users," in Workshop on Mobile Computing Systems and Applications, Santa Cruz, USA, Dec. 1994. http://www2.parc.com/csl/projects/bayou/.
14. R.A. Dirckze and L. Gruenwald, "A toggle transaction management technique for mobile multidatabases," in Int. Conf. on Information and Knowledge Management (CIKM), Bethesda, USA, Nov. 1998.
15. R.A. Dirckze and L. Gruenwald, "A pre-serialization transaction management technique for mobile multidatabases," Mobile Networks and Applications (MONET), vol. 5, no. 4, 2000.
16. M.H. Dunham, A. Helal, and S. Balakrishnan, "A mobile transaction model that captures both the data and the movement behavior," ACM/Baltzer Journal on Special Topics in Mobile Networks and Applications, vol. 2, no. 2, 1997.
17. M.H. Dunham and V. Kumar, "Defining location data dependency, transaction mobility and commitment," Technical Report 98-CSE-01, Southern Methodist University, Dallas, USA, Feb. 1998.
18. M.H. Dunham and V. Kumar, "Location dependent data and its management in mobile databases," in Int. DEXA Workshop on Mobility in Databases and Distributed Systems, Vienna, Austria, Aug. 1998.
19. M.H. Dunham and V. Kumar, "Impact of mobility on transaction management," in Int. Workshop on Data Engineering for Wireless and Mobile Access (MobiDE), Seattle, USA, Aug. 1999.
20. A.K. Elmagarmid, Database Transaction Models for Advanced Applications, Morgan Kaufmann Publishers, 1992.
21. A.K. Elmagarmid, Y. Leu, and M. Rusinkiewics, "A multidatabase transaction model for INTERBASE," in Int. Conf. on Very Large Databases (VLDB), Brisbane, Australia, Aug. 1990.
22. K.P. Eswarn, J. Gray, R.A. Lorie, and I.L. Triger, "The notions of consistency and predicate locks in a database system," Communications of the ACM (CACM), vol. 19, no. 11, 1976.
23. FastObjects by Poet. FastObjects j2 http://www.fastobjects.com/.
24. M. Frodigh, P. Johansson, and P. Larsson, "Wireless ad hoc networking—The art of networking without a network," Ericsson Review, vol. 4, 2000.
25. H. Garcia-Molina, "Using semantic knowledge for transaction processing in a distributed database," ACM Transactions on Database Systems (TODS), vol. 8, no. 2, 1983.

26. H. Garcia-Molina and K. Salem. Sagas, in ACM SIGMOD Conference, San Francisco, USA, May 1987.
27. D. Georgakopoulos, M. Rusinkiewicz, and A.P. Sheth, "Using tickets to enforce the serializability of multidatabase transactions," IEEE Transactions on Knowledge and Data Engineering (TKDE), vol. 6, no. 1, 1993.
28. J. Gray, "Notes on database operating systems," in Advanced Course: Operating Systems, number 60 in LNCS, 1978.
29. J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in ACM SIGMOD Conference, Montreal, Canada, June 1996.
30. J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publisher, 1993.
31. T. Härder and A. Reuter, "Principles of transaction-oriented database recovery," ACM Computing Surveys, vol. 15, no. 4, 1983.
32. R. Hirsch, A. Coratella, M. Felder, and E. Rodrguez, "A framework for analyzing mobile transaction models," Journal of Database Management (JDM), vol. 12, no. 3, 2001.
33. L.B. Huston and P. Honeyman, "Peephole log Optimization," in IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, USA, Dec. 1994.
34. IBM Software Products, DB2 Everyplace, http://www-3.ibm.com/software/data/db2/everyplace/.
35. IBM Software Products, IBM Cloudscape, http://www-3.ibm.com/software/data/cloudscape/.
36. IBM Software Products, WebSphere Everyplace Access, http://www-3.ibm.com/software/pervasive/products/mobile_apps/ws_everyplace_access.shtml.
37. IETF, Mobile Ad-hoc Networks (manet), http://www.ietf.org/html.charters/manet-charter.html.
38. T. Imielinski and B.R. Badrinath, "Wireless mobile computing: Challenges in data management," Communications of the ACM (CACM), vol. 3, no. 10, 1994.
39. J. Jing, O. Bukhres, and A.K. Elmagarmid, "Distributed lock management for mobile transactions," in Int. Conf. on Distributed Computing Systems (ICDCS), Vancouver, Canada, May 1995.
40. J. Jing, A.S. Helal, and A.K. Elmagarmid, "Client-server computing in mobile environments," ACM Computing Surveys, vol. 31, no. 2, 1999.
41. G. Jomier and A. Doucet (Eds.), "Chapter "Bases de Donnes et Mobilit" in Bases de donnes et Internet: Modles, langages et systme," informatique et systmes d'information, Hermes Science Publications, 2001.
42. P.J. Keleher and U. etintemel, "Consistency management in deno," Mobile Networks and Applications (MONET), vol. 5, no. 4, 2000. http://www.cs.umd.edu/projects/deno/.
43. B. Kemme and G. Alonso, "Don't be lazy, be consistent: Postgres-R, a new way to implement database replication," in Int. Conf. on Very Large Databases (VLDB), Cairo, Egypt, Sept. 2000.
44. J.J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," ACM Transactions on Computer Systems (TOCS), vol. 10, no. 1, 1992.
45. K. Ku and Y. Kim, "Moflex transaction model for mobile heterogeneous multidatabase systems," In IEEE Workshop on Research Issues in Data Engineering, San Diego, USA, Feb. 2000.
46. V. Kumar, "A Timeout-based mobile transaction commitment protocol," in ADBIS-DASFAA Symp. on Advances in Databases and Information Systems, volume 1884 of *LNCS*, Prague, Czech Republic, Sept. 2000.
47. V. Kumar, N. Prabhu, M.H. Dunham, and A.Y. Seydim, "TCOT- A timeout-based mobile transaction commitment protocol," IEEE Transactions on Computers, vol. 51, no. 10, 2002.
48. H.T. Kung and J.T. Robinson, "On optimistic methods for concurrency control," in Int. Conf. on Very Large Databases (VLDB), Rio de Janeiro, Brazil, Oct. 1979.
49. M. Lee and S. Helal, "HiCoMo: High Commit Mobile Transactions," Kluwer Academic Publishers Distributed and Parallel Databases (DAPD), vol. 11, no. 1, 2002.
50. Q. Lu and M. Satynarayanan, "Isolation-only transactions for mobile computing," ACM Operating Systems Review, vol. 28, no. 2, 1994.
51. Q. Lu and M. Satynarayanan, "Improving data consistency in mobile computing using isolation-only transactions," in IEEE HotOS Topics Workshop, Orcas Island, USA, May 1995.
52. S.K. Madria and B. Bhargava, "A transaction model for improving data availability in mobile computing," Kluwer Academic Publishers Distributed and Parallel Databases (DAPD), vol. 10, no. 2, 2001.
53. McObject Solutions. eXtremeDb http://www.mcobject.com/extremedb.htm.

54. K.A. Momin and K.Vidyasankar, "Flexible integration of optimistic and pessimistic concurrency control in mobile environments," in ADBIS-DASFAA Symp. on Advances in Databases and Information Systems, volume 1884 of *LNCS*, Prague, Czech Republic, Sept. 2000.
55. J.E.B. Moss, "Nested transactions: An approach to reliable computing," PhD thesis, Massachusetts Institute of Technology, Massachusetts, USA, 1981.
56. Oracle Corporation, Oracle9i Lite: The Internet Platform For Mobile Computing, http://otn.oracle.com/products/lite/.
57. Oracle Corporation, Oracle9iAS Wireless, http://otn.oracle.com/products/iaswe/.
58. T. Ozsu and P. Valduriez, Principles of Distributed Database Systems, Prentice Hall, 2nd ed., 1999.
59. E. Pitoura and B. Bhargava, "Building information systems for mobile environments," in Int. Conf. on Information and Knowledge Management (CIKM), Gaithersburg, USA, Nov. 1994.
60. E. Pitoura and B. Bhargava, "Maintaining consistency of data in mobile distributed environment," in Int. Conf. on Distributed Computing Systems (ICDCS), Vancouver Canada, May 1995.
61. E. Pitoura and B. Bhargava "Data consistency in intermittently connected distributed systems," IEEE Transactions on Knowledge and Data Engineering (TKDE), vol. 11, no. 6, 1999.
62. E. Pitoura and P.K. Chrisanthis, "Scalable processing of read-only transactions in broadcast push," in Int. Conf. on Distributed Computing Systems (ICDCS), Austin, USA, June 1999.
63. E. Pitoura, P.K. Chrisanthis, and K. Ramamritham, "Characterizing the temporal and semantic coherency of broadcast-based data dissemination," in Int. Conf. on Database Theory (ICDT), volume 2572 of *LNCS*, Siena, Italy, Jan. 2003.
64. E. Pitoura and G. Samaras, Data Management for Mobile Computing, Kluwer Academic Publishers, 1998.
65. PointBase Java Databases, PointBase http://www.pointbase.com.
66. A. Popovici and G. Alonso, "Ad-hoc transactions for mobile services," in VLDB Workshop on Technologies for E-Services, Hong-Kong, China, August 2002.
67. C. Pu, G. Kaiser, and N. Hutchinson, "Split transactions for open-ended activities," in Int. Conf. on Very Large Databases (VLDB), Los Angeles, USA, Sept. 1988.
68. K. Ramamritham and P.K. Chrysanthis, "A taxonomy of correctness criteria in database applications," Very Large Databases (VLDB) Journal, vol. 5, no. 1, 1996.
69. K. Ramamritham and P.K. Chrysanthis, "Advances in concurrency control and transaction processing," IEEE Computer Society Press, 1996.
70. M. Satyanarayanan, "Mobile information access," IEEE Personal Communications, vol. 3, no. 1, 1996.
71. M. Satynarayanan, J. Kistler, P. Kumar, E. Okasaki, H. Siegel, and C. Steere, "Coda: A highly available file system for distributed workstation environment," IEEE Transactions on Computers, vol. 39, no. 4, 1990.
72. P. Serrano-Alvarado, "Defining an adaptable mobile transaction service," in Int. EDBT Workshop on Young Researchers Workshop, number 2490 in *LNCS*, Prague, Czech Republic, March 2002.
73. P. Serrano-Alvarado, C. Roncancio, M. Adiba, and C. Labbé, "Environment awareness in adaptable mobile transactions," in Journes de Bases de Donnes Avances (BDA), Lyon, France, Oct. 2003.
74. P. Serrano-Alvarado, C.L. Roncancio, and M. Adiba, "Analyzing mobile transactions support for DBMS," in Int. DEXA Workshop on Mobility in Databases and Distributed Systems (MDDS), Munich, Germany, Sept. 2001.
75. Sybase Inc. Mobile and Wireless, http://www.sybase.com/products/mobilewireless/.
76. R. Vingralek, "GnatDb: A small-footprint, secure database system," in Int. Conf. on Very Large Databases (VLDB), Hong Kong, China, Aug. 2002.
77. G.D. Walborn and P.K. Chrysanthis, "PRO-MOTION: Management of mobile transactions," in ACM Symp. on Applied Computing, San Jose, USA, March 1997.
78. G.D. Walborn and P.K. Chrysanthis, "Transaction processing in PRO-MOTION," in ACM Symp. on Applied Computing, San Antonio, USA, Feb. 1999.
79. G.D. Walborn and Panos K. Chrysanthis, "Supporting semantics-based transaction processing in mobile database applications," in Symp. on Reliable Distributed Systems (SRDS), Bad Neuenahr, Germany, Sept. 1995.
80. L.H. Yeo and A. Zaslavsky, "Submission of transactions from mobile workstations in a cooperative multidatabase processing environment," in Int. Conf. on Distributed Computing Systems (ICDCS), Poznan, Poland, June 1994.