Chapter 1

Fault-Tolerance and Recovery in Mobile Computing Systems

Elisa Bertino, Elena Pagani, Gian Paolo Rossi

1.1 INTRODUCTION

Through wireless networks, mobile personal machines have the ability to access data and services that can be located on both mobile and wired servers. Unlike wired hosts, mobile hosts can be temporarily unreachable as a consequence of their moving across different cells, their energy limitation or unavailable wireless channels. Mobility forces mobile hosts to alternate connected and disconnected work. When connected, they perform personal communications and access shared data and services; when disconnected, they can process locally cached data objects.

As for wired networks, data replication is the key element to ensure high data availability and to increase performance. However, disconnected work and the uncertainties of the underlying wireless network introduce new challenging issues that have been recently discussed in literature. There are three main aspects that we wish to discuss in this chapter:

- how to provide a fault-tolerant architecture that addresses data access and management despite mobility and disconnected work;
- how to manage data replication to ensure data consistency, integrity, durability, according to the application requirements;
- to which extent the general requirement of network independency can be met or otherwise the application awareness of mobility can be effectively exploited to provide the level of quality of service needed.

In this chapter, we investigate how the problem of managing a distributed database and guaranteeing data consistency is affected by the characteristics of the mobile setting. We discuss the impact of mobility and disconnections on fault-tolerance and recovery. We investigate how fault-tolerance can be ensured, by analyzing some of the algorithms proposed for database management in mobile systems.

This chapter is organized as follows. In section 1.2, we describe the mobile environment and introduce data management issues in such an environment. In section 1.3 we present the system architecture to which we refer in the remainder of the discussion, and in section 1.4 we characterize mobile applications. In section 1.5 we investigate how fault-tolerance is affected in mobile environments and how the ACID properties can be re-defined to guarantee data correctness in a mobile setting. In sections 1.6 and 1.7 we discuss some of the approaches proposed for managing distributed databases. Finally, in section 1.8 we report some performance evaluation results concerning some of the described algorithms.

1.2 DISTRIBUTED SYSTEMS WITH MOBILE HOSTS

Unlike computer networks with fixed stations (FHs), a mobile host (MH) can retain its network connection even *while* moving. This is possible because of the use of different network technologies such as radio links, satellite networks or infrared links [35, 36], that do not impose any physical constraint to the hosts; that is, they are *wireless*.

Wireless networks may be classified in *single-hop* [24, 23, 3, 13, 27, 28] and *multi-hop* [12, 11]; in the latter case, all the machines in the system are mobile, while in the former, both mobile and fixed stations are involved.

In the sequel, we restrict our attention almost only to single-hop systems, as they are the most considered in literature.

Single-hop networks are organized as shown in Figure 1.1: some of the fixed hosts, denoted as *Mobile Support Stations* (MSSs) [24, 1, 2, 4] are equipped with a wireless interface; they support communication among both the MHs that reside in a *cell*, and among the MHs in different cells. The cell is the area in which the signal generated by the MSS can be received by the MHs. The messages generated by a given MSS are broadcast within the cell. The MHs filter the messages according to their destination address; on the contrary, a MH can communicate with another MH of the same cell only by sending its message to the cell MSS that executes the broadcast. FHs and MSSs are connected through a *wired* network, whose topology is static and used to support the communication amongst cells. Because of movements, the topology of the wireless network may change over time.

The diameter of the cells may vary according to the wireless technology: e.g. it spans from a few meters for infrared technology, to 1 or 2 miles for radio or single-hop satellite networks [36]. Moreover, the technology also affects the available bandwidth: LANs that use infrared technology have transfer rates of the order of 1-2 Mbps, (up to 100 Mbps in the recent experiments [36]); on the contrary,

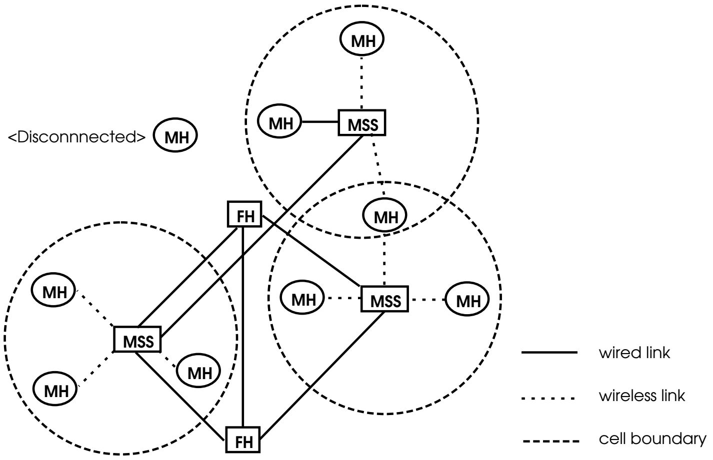

Figure 1.1. Example of a single-hop network

WANs have poorer performance, as they usually provide bandwidths in the range 14.4-64 Kbps. Wireless networks that offer around 100-Kbps services are under development [36]. Finally, wireless networks are supposed to be less reliable than the wired ones: it has been estimated that the failure rate will increase at least of one order of magnitude with respect to the current wired networks.

Cells may overlap, hence a MH may be contemporarily in more than one cell; although it refers to only one MSS a time. In most systems, MHs choose their current MSS according to the highest signal they receive [13, 2]. When the MH_m moves from MSS_1 to MSS_2 a *handoff* (or *handover*) procedure [24, 4] is activated between MSS_1 and MSS_2 to transfer the state information about MH_m to MSS_2. A *mobility assumption* [1, 2] is required to ensure system liveness: MH_m resides in a cell at least for the time sufficient to complete the handoff procedure and to allow the MSS to deliver to MH_m at least one of the messages that were still pending at MSS_1; this way, we guarantee that messages do not starve.

In their wandering MHs could move to places that are out of the cell *coverage*, that is, become *disconnected*. This depends on the system capability to cover a given geographical area. In the U.S. [36] the wireless service with the broadest coverage is Ardis, that reaches more than the 80% of the U.S. population and spans around the 90% of the metropolitan areas and the $30 - 40\%$ of the rest of the country; as a counterpart, Ardis has a very low bandwidth; it offers 4.8-Kbps connections. Other wireless networks exist but, because of the current lack of standards, it is not possible to exploit the services of different wireless networks to gain a greater coverage.

Disconnections can also occur because of several events; e.g. the MH may exhaust its battery charge, it can be lost or it can crash.

MHs can be classified as either *dumb terminals* or *walkstations* [24]. The former ones are diskless hosts (such as, for instance, palmtops) with reduced memory and computing capabilities. They can receive from the wireless network, but they are not able to send messages. Walkstations are comparable to classical workstations and can both receive and send messages on the wireless network. We will focus our considerations on this latter type of MHs.

Despite their computing resources, MHs are mainly constrained by the short lifetime of batteries [8] that are heavily affected by the communications over wireless channels. To reduce the energy waste, MHs enter *doze mode* when they are not involved in sending or receiving packets. A doze MH only has the network interface active, which is able to filter the messages broadcast in the cell on the basis of the destination address. If a message is observed that is addressed to the MH, the system is awaked to revert to the normal operation mode.

The described system behaviours impact on the design of distributed applications. To our purposes the most relevant are:

- the high failure rate requires to address both the fault-tolerance and the recovery issues;
- the energy saving argument generates some new constraints that must be considered while designing the services to support distributed applications.

1.3 SYSTEM ARCHITECTURE

Figure 1.2 helps us to identify the main functional modules that compose a MH architecture [29]. The hardware interface provides the physical access to the network and also filters the messages broadcast in the cell according to their destination address.

<table>
<tr><td colspan="2">Mobile applications</td></tr>
<tr><td colspan="2">Data/Service/Resource mobility</td></tr>
<tr><td>Mobile transport protocol (Mobile–TCP)</td><td>Multicast transport protocol</td></tr>
<tr><td colspan="2">Mobile–IP/Handoff procedure</td></tr>
<tr><td colspan="2">Hardware wireless interface</td></tr>
</table>

Figure 1.2. Reference architecture of a mobile host

On top of it the Network layer, e.g. through the Mobile-IP [31] protocol, provides transparent addressing of MHs and executes the handoff procedure [6, 38].

Communication over the wireless network is unreliable, i.e. packets are lost, corrupted or duplicated, and the transmission delay is highly variable due to different wireless technologies and load conditions. Hence, a wireless network may be considered as an asynchronous system; this implies the unpredictable duration of the transaction processing. Transport layer masks network uncertainties to upper layers and provides some sort of reliable, point-to-point or multicast [1, 2] channels amongst MHs. Certain multicast transport channels (e.g. [1, 2]) can ensure FIFO order in the message delivery.

The higher layers provide the value-added services to directly support the application communication requirements. These services mainly address the management of the data objects and files in the presence of mobility. They are also responsible for negotiating with MSSs the quality of the service according to both the user requirements and the services actually supplied by the wireless network [29, 7, 13, 15, 33, 28, 22].

The problem of *locating* MHs, i.e. of knowing their current position to allow the routing of the messages has received great attention [5, 25] and is emphasized by the trend of reducing the cells size to improve the communication bandwidth. Location service is architecturally located within the network layer, although some interesting evolution of the basic location service are oriented to allow their direct use by mobile applications (see, for instance, [41]).

1.4 MOBILE APPLICATIONS

Applications that run on mobile hosts are likely to have different requirements with respect to those designed for traditional environments. Most users will use MHs for personal communications (e.g. e-mail, around 25%) and for mobile office activities (around 45%) [36]. The latter possibility implies the ability of porting existing applications on MHs and of allowing them to access and share remote data objects.

In [24, 23] a first attemp to classify mobile applications has been introduced based on the locality of the data the application accesses. In *vertical* applications the users access the data within a specific cell and the access is denied to users that are out of that cell. For instance, data concerning the availability of parking places in that cell, the position of the nearest doctor or the personal identities of the other users in the cell. On the contrary, *horizontal* applications handle data that span over users being distributed on the whole system; typically, they are applications whose users cooperate towards a common task, in spite of their movements, or multimedia applications such as conferencing.

The nature of the applications impacts on the pattern of access in reading and writing the data; in particular, in [24, 23] the following classes of data have been identified:

Private data: they are maintained, accessed and managed by a single user, the owner; no other users may access the data.

Public data: one user may update them, while all the users of the system can read them. Consider for instance applications such as weather forecast, news bulletins or broadcast of financial data. Another important kind of information in this category is *location* data [41], that is, data concerning the identity of the cell in which a MH currently resides. In [41] data have been further classified into three categories according to their semantics, that reflects the frequency of their updating: (*i*) *terminal mobility* data, that concern the location of the host; (*ii*) *personal mobility* data, that concern the users identity and are used for the user authentication, and (*iii*) *service mobility* data that describe the users' profiles, regarding, for instance, the customization of the applications they use or the subscribed services.

Shared data: they are accessed both in read and write by a group of users cooperating to a common task (e.g. cooperative workgroup) or managing multiple copies of the data to achieve availability and reliability.

While public data are mainly managed by vertical applications, the use of shared data in the framework of horizontal applications introduces a general and complete range of fault-tolerance and recovery issues that mainly concern the topics of this chapter. In this work, we will consider mainly this setting.

To ensure the service availability and to improve the performance, shared data can be *replicated*. Copies may be located both on fixed and mobile stations. Mobility introduces new challenging issues in the design of the mechanisms that guarantee data consistency and integrity. The *scalability* of these mechanisms over a possibly large amount of MHs is also an important issue.

1.5 FAULT-TOLERANCE IN MOBILE DATABASE PLATFORMS

As the mobile setting highly differs from the fixed setting, it is necessary to redefine what a failure is, and what "fault-tolerance" means in this new context.

In general, a system is fault-tolerant if it guarantees to behave correctly with respect to its service specification despite malfunctions; in the case of database systems, correctness is usually defined in terms of the *ACID properties*. In this section, we explore the approaches to fault-tolerance in database management and show some examples of normal MHs behaviours that may be misinterpreted as failures. We discuss the impact of these behaviours on the correct operation of the system and show how fault-tolerance may be redefined according to these considerations, and how to achieve it. In the following, we do not consider the failures on the wireless network because their detection and recovery are responsibility of the transport protocol. A reliable transport service is observed at the interface with the transport protocol (see section 1.2). The services we consider throughout this chapter are built on top of such reliable transport protocol.

1.5.1 Transaction execution in mobile database systems

The characteristics of MHs introduce new fault-tolerance issues in transaction management. Among these issues the capability of tolerating the disappearance of MHs from the cells is of primary concern because of mobility and disconnections. Whether the MHs store the entire database or part of it, and actively participate in the management of the database is a design choice that impacts on the effects that failures may have.

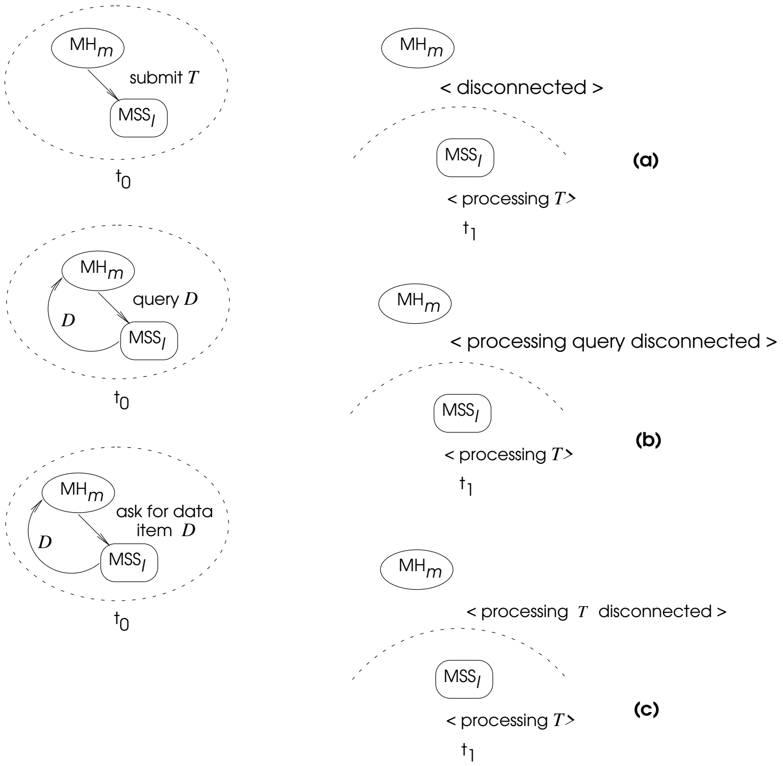

Figure 1.3. Management of mobile databases. (a) The MH$_m$ submits a transaction $\mathcal{T}$ according to the transaction proxy approach. (b) The MH$_m$ submits a query on the data D according to the read-only transaction approach. (c) The MH$_m$ submits a transaction $\mathcal{T}$ according to the weak transaction approach.

Most approaches assume that copies of data on MHs are *secondary* copies, while *primary* copies are maintained at FHs and MSSs. As we will see in section 1.6, different approaches may be used that give more or less autonomy to MHs in operating upon the database. Almost all these approaches, however, force MHs to perform periodic checkpoints and to maintain their backups on FHs. The adopted approaches may be classified as follows:

transaction proxy: the MHs do no execute any computation, but instead ask the MSSs to execute transactions on their behalf [13, 44, 27]. Therefore, the MSSs always hold the consistent database, while MHs do not need to execute any update action on the data objects nor to keep any data object in their caches (Figure 1.3(a));

read-only transactions: MHs only cache data objects for queries, while updates are performed as in the above case (Figure 1.3(b));

weak transactions: besides performing queries on cached data, MHs may update data objects in their caches even while disconnected [28, 30, 7, 40] (Figure 1.3(c)). In this case, they must *stabilize* their updates as soon as they reconnect, that is, they have to globally commit the updates in order to reestablish consistency and to guarantee durability. For the purpose of stabilizing the disconnected transactions or *undoing* them in the event of an abort, a *log* is maintained in secondary storage at the MH, recording the actions executed by uncommitted transactions [28, 30]. The log of each transaction is sent to the MSS upon reconnection, so that the MSS can re-execute the transaction on its primary copy, to verify whether it can safely commit.

It should be evident that in the case of the weak transaction approach applications have to deal with more difficult fault-tolerance problems than in the transaction proxy approach, because of the maintenance of data objects stored by the MHs. In the weak transaction approach, recovery mechanisms must be designed by properly taking into account the scarce availability of storage resources of walkstations.

1.5.2 Impact of mobility on transaction correctness

In section 1.2, we have shown that a mobile environment is characterized by hosts that can temporarily be unreachable, because of entering doze mode, disconnecting or moving to uncovered zones, and by the intrinsic asynchrony of the underlying network environment. In this section we discuss how these features may lead to the violation of ACID properties or may jeopardize the liveness of the system even in the absence of failures.

Figure 1.4 shows two cases. In the first case (Figure 1.4(a)) MH_m caches a set D of data objects while being in cell l, then it disconnects and continues to process its transactions while being disconnected. When it later reconnects to the same or to a different MSS, its copy of the data is inconsistent with respect to the one held at MSSs. The same problem also arises when the MH_m caches data for read-only transactions because of the updates carried out by other FHs or MHs, while MH_m is unreachable.

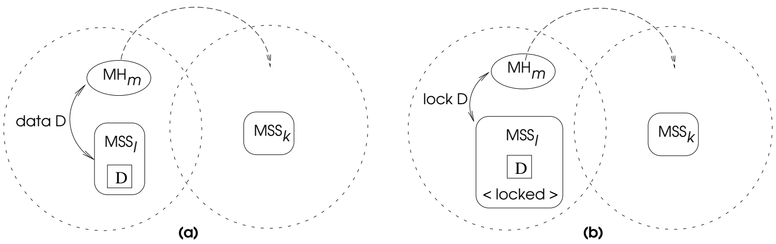

Figure 1.4. Example of possibly incorrect behaviours in the case of movement of an MH

To solve this problem, MH_m can lock the data objects D at the MSS_l site while executing local transactions, thus preventing concurrent execution of other transactions originating from the MSS_l or other MHs (Figure 1.4(b)). If MH_m disconnects or cannot be reached for a long time, it can be suspected of having failed. If MSS_l maintains the lock, problems arise in the event of actual crashes or long lived disconnections. If the MSS_l releases the lock after a timeout the problem arises of inconsistent copies (Figure 1.4(a)).

The above problems arise because of the difficulty in distinguishing a temporaneous unreachability from a crash. It is, however, possible to distinguish between planned and sudden disconnections [4]. The former are predictable. When the MH becomes doze, or it disconnects either to recharge the batteries or to save power, some safety actions can be performed to tolerate such a temporary disconnection. For instance, the pending transactions can be moved to the destination MSS where an *agent* can execute them on behalf of the mobile application to allow the MH to obtain the results upon reconnection, as proposed in [13]. Another approach allows MHs to *prefetch* in the cache the data they require while being disconnected. Appropriate algorithms are designed to re-establish consistency among the existing copies of the data when the MH reconnects.

1.5.3 Transaction correctness in mobile computing systems

The ability of working in a mobile context must coexist with the possibility that even normal system conditions may lead to the violation of the database correctness. As a consequence, the efforts to achieve fault-tolerance have been addressed to redesign the notion of correctness rather than to redefine the notion of failure. A number of alternative definitions of the ACID properties have been proposed [30, 32, 21, 15] that weaken one or more of the properties. In general, their goal is to guarantee the MHs a certain degree of autonomy in transaction processing during disconnections, and to preserve the (modified) system correctness by allowing bounded inconsistencies among the data copies.

In the following, we describe how each property has been redefined in some proposals in the scientific literature. Usually, only one property is considered at a time. The weakening of a given property however may impact the other ACID properties.

ATOMICITY PROPERTY The first step towards fault-tolerance in mobile systems is allowing MHs to submit "*pieces*" of transactions in different cells according to their movements. Several alternative methods are described in [21]. This approach weakens the classical formulation of atomicity and requires the ability of breaking a transaction so that *subtransactions* can be concurrently executed and interleaved with subtransactions of other transactions while at the same time guaranteeing the other ACID properties. These mobile transaction models are based on extended transaction models developed for long duration transactions, such as Open Nested Transactions (ONT) [16] and Saga Transactions [18]. For instance, in the ONT model the abort of one or more subtransactions does not necessarily imply the abort of the entire transaction. Hence, when a transaction $\mathcal{T}$ commits only some of its operations may have been actually executed.

The decomposition of a transaction in subtransactions can be performed according to different principles and at different levels of granularity. Different approaches to decomposition have been proposed for each of the three transaction models discussed in section 1.5.1. In the case of transaction proxies and read-only transactions, transactions may either be submitted as a whole at a unique MSS [13, 44], or they may be *split* during processing (Kangaroo model, [21]) following the movements of the MH that submitted the transaction. In the latter case communication costs are reduced by relocating computations as near to the MH as possible.

In the case of weak transactions, a transaction $\mathcal{T}$ is decomposed into mutually independent subtransactions. This decomposition ensures that the subtransactions of a transaction $\mathcal{T}$ can be *concurrently* processed at the different MSSs to which they have been submitted, and that their execution order does not impact on the successful commit of $\mathcal{T}$. Independence may be guaranteed for instance according to Bernstein's conditions [9]. Each subtransaction $\mathcal{S}_i$ has a write set W_i and a read set R_i. Every two subtransactions $\mathcal{S}_i$ and $\mathcal{S}_j$ of a transaction $\mathcal{T}$ satisfy the following conditions:

$$W_i \cap W_j = \emptyset \qquad R_i \cap W_j = \emptyset \qquad W_i \cap R_j = \emptyset$$

that guarantee their independence (Reporting and Co-transaction model [21]). This approach is based on the Split Transaction Model [34]. A run-time support must exist that computes the decomposition by determining an appropriate partition of the read and write sets of the transaction.

Another approach is based on fragmenting a transaction $\mathcal{T}$ so that each of its subtransactions $\mathcal{S}_i$ executes operations that are *commutative* with those of the other subtransactions of $\mathcal{T}$ [21].

Both in the Kangaroo model and in the Reporting and Co-transaction model the handoff procedure must be extended to involve the transfer of information concerning pending transactions generated by the MH. These models imply a redefinition of the other ACID properties. Both isolation and durability are restricted to subtransactions instead than global transactions. In the case of weak transactions the consistency property is also affected, and a mechanism is required to merge copies and re-establish consistency upon reconnection.

CONSISTENCY PROPERTY An approach alternative to that of weakening the Atomicity property consists in the redefinition of the consistency property. Under this approach, the database is considered partitioned in *clusters*, either according to semantics-based criterions (e.g. data objects related by integrity constraints belong to the same cluster) or to location proximity (clustering model [32, 21]). Data in the same cluster must be strictly consistent, while a *bounded* degree of inconsistency is tolerated amongst clusters, according to some definition of consistency. Hence, clustering can for instance support multiversion databases or tolerate divergences between the secondary copy of the data maintained at a MH (that constitutes a cluster) and the primary copies on FHs. MHs are therefore allowed to process transactions while being disconnected. According to this approach, two classes of primitives are used to update data:

- *weak-write* and *weak-read* that modify data only in the local cluster, thus possibly causing inconsistencies with respect to other clusters;
- *strict-write* and *strict-read* that modify data in the global database, thus maintaining consistency.

The above primitives are executed so that operations that work on the same cluster do not conflict. Conflicts are prevented by locking mechanisms. In assigning locks the usual lock compatibility modes are applied. Moreover, to guarantee that weak operations do not observe intermediate results produced by strict operations, strict-write locks and weak-read locks conflict. By contrast, strict-read and weak-write operations are not conflicting operations. The implementation must only guarantee that a strict-read operation reads the value written by the last strict-write operation.

A consequence of the redefinition of the consistency property is a more complex notion of serializability. Strict transactions must serialize with respect to each other according to the one-copy serializability [17]. Moreover, let the *projection* of a strict transaction $\mathcal{T}$ on a MH_m be the subtransaction of $\mathcal{T}$ that operates on the data objects held at the MH_m. Weak transactions processed at MH_m must serialize always according to the one-copy serializability with respect to each other and with respect to the projections on MH_m of the strict transactions.

The degree of inconsistency can be defined for instance in terms of the maximum number of versions of the same data objects that can exist at the same time, or the maximum number of weak operations that can be executed on a copy of a data without being propagated to the other copies [33]. A timestamp can be associated with each data so that locks on that data are automatically released after the expiration of a timeout (time-based consistency model [21]). This way, MHs can operate disconnected for a limited interval of time.

Isolation property Some transaction models have been devised for mobile environments in which the isolation property is not guaranteed, that is, intermediate results of a transaction $\mathcal{T}$ can be observed by other transactions. This is usually a side effect of the relaxation of other ACID properties.

We have observed in the previous paragraph that isolation can be enforced by properly modifying conflict rules amongst locks, in the case of operations that work on data having different degrees of consistency.

By contrast, if a transaction model is adopted such as those mentioned in the discussion on atomicity, the isolation property only holds for subtransactions. If a subtransaction $\mathcal{S}_j$ of a transaction $\mathcal{T}_j$ is processed at a MSS after that a subtransaction $\mathcal{S}_i$ of a transaction $\mathcal{T}_i$ has been processed at the same MSS, then $\mathcal{S}_j$ can observe the result of $\mathcal{S}_i$, that is, an intermediate result of $\mathcal{T}_i$. Helal et al. observe in [21] that the sharing of partial results amongst transactions, e.g. by means of *Reporting Transactions*, may be desirable for some applications. On the other hand, enforcing isolation property is probably expensive in that it may severely restrict concurrency amongst transactions.

DURABILITY PROPERTY Durability of committed transactions is mainly affected by the possibility MHs have of autonomously operating on data. In some transaction models MHs can execute transactions on locally stored data even while being disconnected (Reporting and Co-transaction model, Clustering model [21]). If a MH_m operating in disconnected mode fails before *stabilizing* the results of its committed transactions on the primary copies of the database, these results could be never recovered (for instance, in the case of a media failure).

To ensure fault-tolerance, in the CODA file system [28, 30] the durability property is relaxed by providing two types of transactions and two degrees of commitment. *First class transactions* are those executed by either connected MHs or users on FHs, while *second class transactions* are those processed by disconnected MHs.

A disconnected MH can only commit a transaction *locally*, if this transaction does not conflict with other transactions executed on the *same* host, while the host is disconnected. Upon reconnection the transactions are *globally* committed, unless they conflict with already committed first class transactions or second class transactions executed on different MHs.

Hence, first class transactions have one level of commitment:

- a first class transaction can commit if it is serializable with respect to all the transactions previously committed.

By contrast, second class transactions are subject to two levels of commitment:

- *local commitment*: the transaction can commit if it is serializable with respect to all the previously committed second class transactions executed on the same host;

- *global commitment*: the transaction can commit if it is serializable with respect to all the committed transactions in the system.

Two levels of commitment have also been adopted for the weak transactions described in the Consistency paragraph [32]. The global commitment allows to detect possible inconsistencies caused by weak transactions upon reconnection. A locally committed transaction can however globally abort. The durability of locally committed transactions is *not* guaranteed until these transactions globally commit.

1.5.4 Recovery in mobile databases

Transaction recovery deals with the capability of ensuring *failure atomicity* [14]. It concerns the durability and the atomicity properties. Recovery mechanisms guarantee that these properties are satisfied in spite of failures.

As in the most proposals found in the current literature, we assume that the fixed network is reliable. We therefore focus on the problem of recovering MHs. As in the case of fault-tolerance, we must understand which situations require recovery.

Moreover, we investigate how recovery could be achieved according to the limited computing and storage resources of the MHs.

According to [20] we may classify failures in three categories:

transaction failures: a transaction may abort because the MH was disconnected, as in the read-only transactions and weak transaction approach. Upon reconnection the invalidation of its cache is communicated to the host, or a conflict is raised between the updates of the transaction and the updates of other, possibly already committed, transactions;

site failures: the MH crashes but the content of its permanent storage is not lost;

media failures: the loss of part or all the secondary storage holding the database can occur.

Recovery makes in general use of a *log* file recording information on the operations executed by both committed and still uncommitted transactions, the last safe state and what else is needed to rebuild a consistent database in case of failures. This information is used during recovery to *undo* the partially executed transactions (atomicity property) and to *redo* the committed transactions (durability property) based on the last safe state.

As we have seen in the previous section, atomicity in mobile environments is affected by failures when a MH_m either (i) processes transactions while being disconnected, or (ii) submits subtransactions. In the latter case, a subtransaction $\mathcal{S}$ may have to be undone whose results have been observed by other subtransactions, possibly belonging to an already committed transaction. In [21] this situation is dealt with by executing *compensating* transactions that *semantically* undo the effects of $\mathcal{S}$ (Saga model [21]). This solution is not however always viable because some operations are inherently non-compensatable.

If the MH_m carries some data and autonomously operates upon them, as in case (i), other MHs could concurrently update the same data. Hence, if MH_m recovers from a failure, the undo of pending transactions produces an obsolete version of the database.

For this reason, under the most widely adopted solution the recovered MHs refetch the updated version of the data objects to report their database view to a consistent state.

The durability property is ensured only on fixed stations. The problem is to determine when the data maintained at MHs need to be recovered, and the most appropriate recovery technique according to the adopted transaction management policy. With respect to the classification we have presented in Section 1.5.1, the recovery mechanisms are as follows:

Transaction proxy: as MHs do not maintain any data object, none of the above failure modes needs a MH recovery.

Read-only transactions: the MH_m cache may become out of date during disconnections, movements or crashes. Usually, data have a version number

[28] or timestamp [21] so that invalid caches can be detected upon reconnection. When MH_m reconnects, it is sent an invalidation message from the MSS (*server callback*). Such a message could result in query aborts. It is up to the MH_m to refetch the invalidated data and to re-execute the aborted queries. A different approach is taken in *data-broadcasting* algorithms [26, 10], where MSSs broadcast either the whole database or the more frequently accessed data objects and MHs autonomously keep up to date. We further discuss these algorithms in section 1.6.

Weak transactions: if the disconnected transactions cannot globally commit, according to the definitions we gave in Section 1.5.3, a transaction failure occurs. The problem here is how to make durable the locally committed (but globally aborted) transactions. The recovery procedures that can be adopted range from the automated refetch of the updated data and reexecution of the globally aborted transaction, to the user notification, to the execution of an application-dependent algorithm [30]. Such an algorithm has the purpose of understanding whether the globally aborted transaction can be dropped, or only a part of it can be reexecuted, or one of the two previous solutions can be adopted, according to the application semantics [30]. The redo is performed according to the recorded log file. After the redo is completed, a global commit is tried again. Site failures do not affect either the redo of globally aborted transactions or the global commit. By contrast, media failures that cause the loss of the log file are unrecoverable. Transactions not yet globally committed are lost, and the failure has to be reported to the application.

In section 1.6 we describe some of the algorithms that have been proposed and the recovery mechanisms they implement. An important question is that in mobile systems, because of mobility, disconnections, and higher failure rates than in static environments, recovery procedures are likely to be executed more frequently. Hence, besides being lightweight in terms of the required MH resources, recovery procedures should support fast transaction restart.

1.6 CLASSES OF SOLUTIONS

We discussed in the previous sections how fault-tolerance and recovery concepts are adapted to a mobile environment. In this section, we describe algorithms for managing replicated data in mobile environments. The aim of this description is to highlight the capability of a given approach to satisfy the above mentioned fault-tolerance requirements.

1.6.1 The "data-broadcasting" approach

The data broadcasting approach is a special case of the *read-only transactions* approach in which MHs can only query data. The database resides on one or more FHs; a MSS holding a copy of the database periodically broadcasts this copy to the MHs in the cell (Figure 1.5). The broadcast database version corresponds to a checkpoint at a given time; updates are performed between two successive broadcasts. ACID properties are guaranteed on FHs and mobile queries cannot cause inconsistencies.

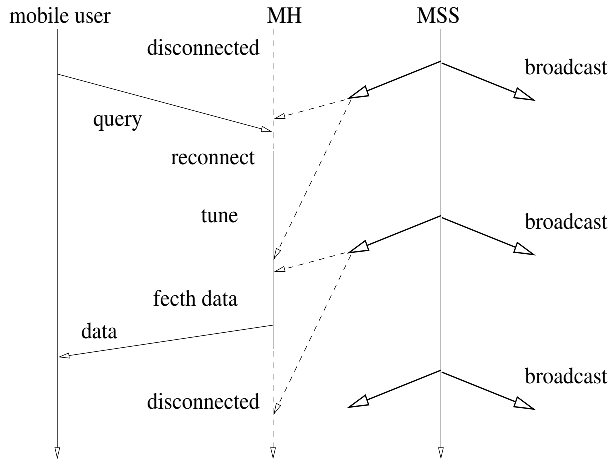

Figure 1.5. Layout of the data-broadcasting approach

This approach requires that a MH stays active by listening to all the incoming messages until it receives the desired data. Some improvements to this approach have been proposed that allow MHs to *tune* in when data are transmitted to save energy. This is achieved by broadcasting only the *hot-spot* data, that is, the most frequently accessed data, and by periodically broadcasting an *index*, or *directory*, of the database. Indexes are interleaved with blocks of data and allow MHs to determine which data will be sent next.

The broadcast of frequently accessed data is based on the principle of *data access skew*, that is, on the hypothesis that data objects are not accessed with the same frequency (80/20 rule) [19]. Less accessed data can be retrieved *on demand*. Explicit requests of data are recorded by MSSs and used to adapt the hot spot composition accordingly.

The reader can find in [26, 10] details on how to interleave index information with data in order to optimize the amount of time a MH has to be connected before receiving the data.

By adopting the data broadcasting approach, MHs cannot suffer of transaction failures. Simple fault-tolerant mechanisms are required to ensure consistency of the data that MHs are likely to cache. The introduction of proper information about the modified data in the index results in an efficient policy to selectively update cached data while a MH is connected [26]. In the event of a disconnection or a crash, the consistency is achieved by refetching the data copies in the MH cache. The periodic broadcast of index and data guarantees that an MH eventually receives the required information despite network packet loss and corruptions.

The main advantage of this approach is simplicity and low complexity at the MHs. These advantages are, however, obtained at the price of low performance and high bandwidth consumption. It is suitable for applications that involve simple MHs, e.g. dumb terminals, accessing public data on FHs.

1.6.2 The "proxy" approach

Under the transaction proxy approach, MHs can generate both queries and transactions. Data are maintained at FHs.

Two methods may be distinguished for managing transactions: (*i*) MHs submit the transactions or subtransactions [13, 44]; (*ii*) MHs submit the requests for read or write-locks on the required data objects [27].

Under the former method, MHs do not maintain data. MHs submit transactions or subtransactions to the MSSs they visit while moving. Fixed stations are in charge of enforcing correctness properties on data by adopting proper fault-tolerant mechanisms. If a MH fails while submitting the query the failure does not affect the database. By contrast, under the latter method, MHs can maintain local data. The locks on the data, however, are recorded and managed at the MSSs.

In [13] the database is assumed to be fully replicated at the MSSs and the MHs can submit transactions to their current MSS. The ISIS system provides the required fault-tolerance within the group of MSSs. The ISIS ABCAST primitive is used to ensure a total order in the delivery of the multicast messages that transport transactions. MH status information are partially replicated over cluster of MSSs centered around the current location of the MH. Both status and location data are considered and managed as the database objects.

In [44] MHs can submit subtransactions and are free to distinguish between *global database* and *local database*, which is locally maintained and accessed. The global database is replicated over the group of MSSs. Data correctness is guaranteed among the copies of the global database. Suppose that a global transaction or subtransaction T_g is executed that precedes a local transaction T_l which satisfies some integrity constraints between global and local databases. If T_l causally precedes another global transaction T_{g1}, the group of MSSs database servers must process T_{g1} after T_g. To enforce this causality in the transaction processing order, global transactions or subtransactions carry a *ticket*, i.e. a *timestamp*.

The method described in [27] is quite similar to the one in [44]. Unlikely that, it assumes that MHs can require read and write locks to "handle" the data according to a revised version of the *optimistic 2-phase-locking*. Before accessing a data item, an MH_m must require the appropriate lock to the current MSS (MSS1); read locks are immediately granted, while write locks are postponed at commit time.

Because of mobility, locks may be requested to different MSSs. Lock information could be transmitted together with the MH state during the handoff procedure. To improve performance, however, read locks and unlocks are maintained at the MSSs to which they have been required, up to the commit time (Figure 1.6(a)). At

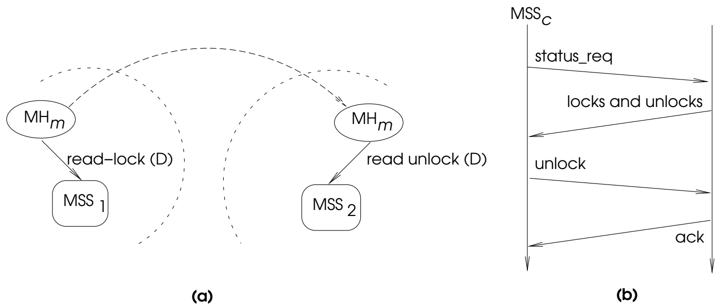

Figure 1.6. Optimistic 2-phase-locking for mobile systems

this point, before granting write locks, the current MSS checks for the existence of conflicting locks on the other MSSs. The procedure is shown in figure 1.6(b) where MSS$_c$ is the current MSS. It sends to the other copies a status request and waits for *all* the replies, reporting the locks and unlocks recorded in those sites. MSS$_c$ tries to *match* each read lock with a corresponding unlock. If it succeeds and no other write lock already exists, it releases all the existing read locks on the other copies and, after receiving their acknowledgements, it grants the write lock to the MH. A copy sending the acknowledgement records at the same time to be write-locked.

An MH failure or disconnection can leave some data unlocked. This problem can be handled by associating a *timeout* with each lock. Upon expiration of the timeout the lock is unilaterally released. The timeout may be specified by the user. Once the timeout is expired, the MSS that drops the lock must inform the other copy holders that the transaction has aborted. The MH is notified upon reconnection, to make its state consistent and to preserve the atomicity property.

Serializability is guaranteed by associating version numbers with the data. Moreover, each copy must record a write-intent lock when it receives a status-request message by MSS$_c$.

The described methods are suitable to manage both public and shared data and can support both vertical and horizontal applications. The algorithm described in [44] also considers the possibility of combining private and global data that have some integrity relationship.

1.6.3 The "disconnected" approach

The disconnected approach has been introduced with the Coda file system [28, 30] and has been adopted in other algorithms to manage either file systems [40, 39] or databases [33]. It uses the weak transaction approach and enforces correctness by relaxing either the isolation property [28, 30] or the consistency property [33, 40, 39], as we described in Section 1.5.3.

The mentioned algorithms assume that the database is fully replicated on the MSSs. MSS copies are considered the *first class replicas*, whose consistency is always guaranteed. The MHs can host *second class replicas* of the database, or of a part of it.

MHs and MSSs can execute transactions on the database according to an *optimistic* concurrency control strategy.

Transactions initiated by connected MHs are executed so that the usual definition of the ACID properties is satisfied. MHs work on the data they have cached up to commit time. In [33] these *strict* transactions are processed guaranteeing one-copy serializability. When a strict transaction commits, all the first and second class copies are updated accordingly, thus automatically guaranteeing cache consistency. In CODA only fixed hosts apply the changes to their copies; hence, data maintained on MHs can become inconsistent. Cache coherence is enforced among connected MHs with a protocol based on *callback* primitives. The MH whose cache has been invalidated or that experiences a cache miss during the transaction processing can obtain the updated data on demand from its MSS.

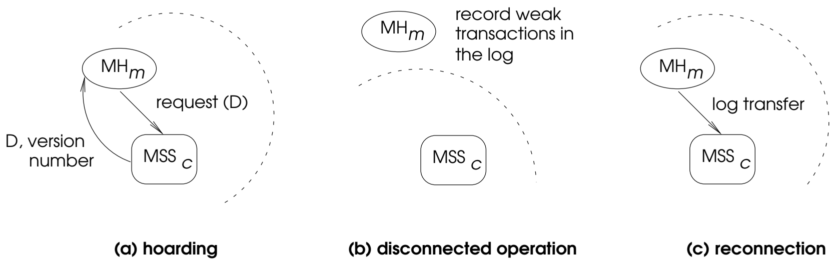

Figure 1.7. Weak transactions in the CODA file system

Disconnected MHs can only rely on the contents of their caches. The problem of cache management in mobile environments has been widely discussed in [42, 43]. To this purpose CODA includes the special-purpose module *Venus*. The Venus module is located at the MH and operates to maintain in cache the most recently used data. While preparing for a disconnection (*hoarding* phase), it can also use user's hints to fetch in the cache the data that are likely to be needed once disconnected (Figure 1.7(a)). Data are tagged with *version numbers*, that are also stored in the cache.

A disconnected (*weak*) transaction $\mathcal{T}$ is processed so that it *locally* serializes with the other weak transactions executed on the same host (section 1.5.3). In the approach proposed in [33], weak transactions can also be executed by connected MHs. Therefore, communication on the wireless network is reduced, but there is the additional requirement that weak transactions cannot observe partial results of strict transactions. Operations executed on the database by disconnected transactions are recorded in a *log file* together with the data version numbers (Figure 1.7(b)). The log file is stored in permanent storage together with the cached data. This allows the MHs to survive long disconnections in spite of the reduced size of the volatile storage. The effects of weak transactions are not permanent until they *globally commit*.

The global commitment is executed when MH_m reconnects (Figure 1.7(c)). The log file is transferred to the current MSS_c, that checks for conflicts by comparing the version numbers recorded in the log with those currently associated with its copy of the data. MSS_c detects conflicts by building and analyzing a *precedence graph* amongst the weak transactions and the previously committed strict transactions. If no conflicts are detected, the MSS_c locks all the data upon which disconnected transactions operated, and *redoes* these transactions according to the trace in the log. A *commit* message is sent to MH_m and the results are propagated to all the connected hosts holding a database copy. Locks on data are released. Otherwise, an *abort* message is sent to MH_m; *cascading* abort only affects pending disconnected transactions executed on MH_m. In CODA a *reply file* is sent together with the abort message, containing the results of the MSS_c attempt and the current state of the database at the MSS_c.

If a transaction aborts during the global commitment phase, three solutions are proposed in [28, 30]: (*i*) the abort is notified to the application; (*ii*) MH_m refetches the updated data and reexecutes the aborted transaction; (*iii*) application-dependent algorithms are executed. These algorithms examine the reply file to decide whether one between solutions (*i*) and (*ii*) is appropriate, or whether the transaction can be only partially redone.

Transaction failures can also be experienced by MHs on local commitment. A disconnected transaction aborts when a cache miss occurs, or the cache overflows as a consequence of either the increase in the log size or the creation of new data. In these cases, the MH has to suspend transaction processing waiting for reconnection.

Since these algorithms are able to tolerate long disconnections of the MHs without either blocking the system or jeopardizing the correctness of the primary copies of the database, the same mechanisms allow them to tolerate both MHs site and media failures.

From the point of view of MH recovery, site failures are not harmful. The transport protocol ensures that the log transfer is performed as an atomic action. Hence, the MH fails either before the transmission be successfully completed or after that. In the former case, the global commitment of pending transactions is executed when MH recovers from the crash. In the latter case, the MSS autonomously proceeds in its processing and the message containing the outcome of the global commitment is delivered to MH when it recovers.

On the contrary, media failures causing the loss of the log and the cached data cannot be recovered and the failure has to be notified to the application.

These algorithms can be executed by MHs having non negligible computing and storage capabilities, such as walkstations. They can be used to manage both public and shared data, and to support both horizontal and vertical applications. In the last case, however, the database cannot always be replicated at all the MSSs, and a MH_m must reconnect in a cell in which the access to the database is supported to stabilize its weak transactions. Finally, since processing is executed locally, both long and interactive transactions are supported for MHs.

1.6.4 The "distributed" approach

This approach adopts the weak transaction model as the previous one. It differs from the disconnected approach in that it supports only strict transactions, that require cooperation amongst the hosts holding a copy of the database. The correctness notion used in this approach is based on the usual definition of the ACID properties. In literature, only [7] describes an algorithm that follows this approach; hence, we refer to it in the following description.

Copies of the database are maintained at both FHs and MHs. Copies are classified in: *core* (primary) ones and *cached* (secondary) ones, according to their consistency degree. This classification is however independent from the type of the host on which a copy resides. It rather depends on whether the host actively participates in the data management or not. Cached copies are maintained at the hosts where applications run that can tolerate inconsistent data. Only queries can be executed on cached copies. On the contrary, hosts requiring consistent data or wishing to generate transactions must belong to the group of core sites. Core nodes periodically generate multicast messages containing the current version of the data, to bring up to date the cached copies. The delivery of these messages must serialize with the queries processed at the cache sites.

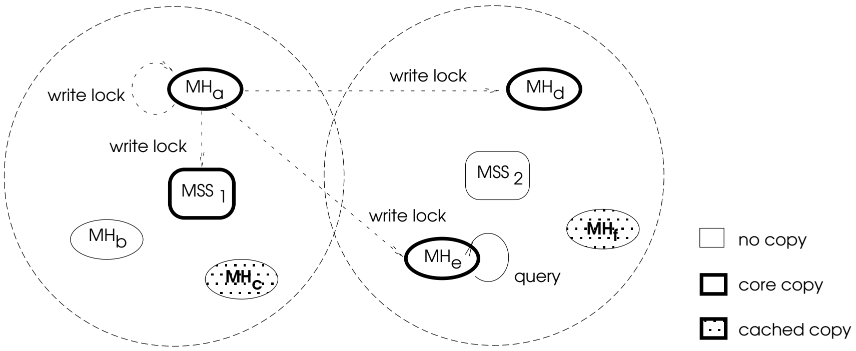

Figure 1.8. Layout of the distributed approach: read-one-write-all method; MH_e makes a query while MH_a requires write-locks for a transaction

Transactions generated by core sites are processed according to existing algorithms used in wired networks; in particular, in [7] the following approaches to transaction processing are considered:

1. **primary/backup:** transactions are sent to a coordinator node (the *primary*) that processes them on its local copy and propagates the result to the other core sites (the *backups*);
2. **read-one-write-all:** the host that generates a transaction $\mathcal{T}$ must lock all the core copies before processing $\mathcal{T}$, while a query can be performed on the local copy when this copy is not locked (Figure 1.8);

3. **general quorums:** a *weight* is assigned to each core copy; quorums are defined for both read and write operations. The host that generates a transaction must lock enough core copies before processing it, so that the sum of their weights equals the appropriate quorum;

4. **majority quorum:** this is a special case of the previous approach. Each core copy has weight 1 and a majority of the weights must be obtained for both read and write operations.

In both approaches (3) and (4) the results of transactions are propagated to all the unlocked core copies. The above approaches guarantee the ACID properties as they do in wired networks.

Note that the previous approaches do not distinguish between FHs and MHs. A mobile station that generates a transaction is required to stay connected up to commit time.

Information on the acceptable quorums is recorded in a *directory*. In [7] the authors suggest that the best technique to manage the directory is to maintain a copy of it at each core site, that is, at the hosts that need to read the directory. The copies of the directory can be updated as they were user data, for example to modify the quorums. Transactions modifying the directory must serialize with the queries to the directory made by transactions accessing the database, according to the one-copy serializability.

The directory also contains information about the membership of the group of core sites. The core group can dinamically change. A MH_m that either voluntarily disconnects or observes a degradation in the wireless communications must renounce to its core copy. It can either become a cached copy or give up. In both cases, MH_m must update the directory to publish the exit of this core copy from the core group. A transaction on the directory is executed by MH_m on its local copy, and only affects the data concerning MH_m (*primary-by-row* update algorithm). Therefore, problems of consistency and concurrency amongst transactions on the directory are avoided. The result is then propagated to the other core sites.

The above procedure allows to manage planned disconnections. By contrast, the algorithm in [7] does *not* tolerate sudden disconnections of core nodes, whichever is the method used to execute transactions. In the primary/backup case the failure of the primary node causes the loss of all the transactions queued at it waiting for processing. With the read-one-write-all method the system is blocked in the evenience of even only one failure. In cases (3) and (4) quorums may become unreachable in consequence of one or more disconnections, thus leaving the system blocked. Sudden disconnections include site and media failures.

Transaction failures occur when either a host contacts the wrong primary site or tries to lock an unacceptable quorum, because it has read an old version of the directory and is still unaware of the last updates. These failures are recovered by reexecuting the aborted transaction.

As for the disconnected approach, the distributed approach is suitable for all mobile applications.

1.7 COMPARISON

In this section we qualitatively compare the characteristics of the algorithms we have described in section 1.6 while we leave some quantitative analysis to be discussed in section 1.8. The analysis concerns:

- features of the services offered to the applications (1.7.1);
- requirements on the underlying system (1.7.1);
- power saving issues (1.7.1).

Fault-tolerance and recovery aspects are separately considered in section 1.7.2.

The main characteristics of the described approaches are summarized in Table 1.1. In the table we report the following items:
- the number of generated wireless messages;
- whether or not MHs can be disconnected during transaction processing;
- whether or not MHs can autonomously process transactions;
- whether or not the approach guarantees fault-tolerance;
- the kind of application for which each approach is more suitable;
- the type of data that are better managed by each approach;
- whether or not the data copies are kept consistent;
- whether or not the approach requires an underlying location service;
- the amount of time a MH has to remain connected and active;
- whether or not transactions may abort as a consequence of conflicts;
- the type of MH more suitable for each approach;
- the communication pattern in which MHs are involved;
- the amount of computation executed on MHs.

Approach	data-broadc.	trans. proxy	disconnected	distributed
Wireless messages	1	1 - 2	2	$O(n)$
Disconnected Work	Yes	Yes	Yes	No
MH's autonomy	No	No	Yes	Yes
Fault-tolerance	Yes	Yes	Yes	No
Applications	Vertical	Horizontal	Horizontal	Horizontal
Data	public	shared	shared	shared
Data consistency	Yes	Yes	No	Yes (core); No (cached)
Location problem	No	No	No	Yes
MH's activity	trans. process.	trans. submiss.	trans. process.	always
Conflicts	No	No	Yes	No
MH's type	dumb/walkst.	dumb/walkst.	walkstation	walkstation
MH's communica.	point-to-point	point-to-point	point-to-point	multicast
MH's computation	low	low	high	high (primary or quorums); low ($\neq$ primary)

Table 1.1. Summary of the described approaches

1.7.1 Qualitative analysis

The approaches proposed to address the problem of data management in distributed systems with MHs widely differ under several respects. Differences arise because of the different system features on which the developers of the various systems have focused their attention, as well as the different mobile application requirements they have considered. In this section we discuss and compare the described approaches.

Service features

The different characteristics of the described approaches impact on the features of the services that each approach can support. Our discussion is based on the consideration that mobile applications have varying requirements and characteristics:

- applications may require that a group of MHs operate tightly coupled or, on the contrary, that they can work disconnected;
- applications may be characterized by short or long and conversational transactions;
- applications may involve several users that simultaneously access the same data (such as, for instance, in cooperative workgroup applications). On the contrary, the concurrency degree amongst users may be low.

If MHs are dumb terminals, they can bear a low degree of **autonomy** because of their reduced computing and storage resources. The choice of moving transaction processing to MSSs is justified by the computational costs of the protocols. Hence, the data-broadcasting and the transaction proxy approaches must be adopted. By contrast, the application requirement of autonomous work can be satisfied by fully exploiting the potential of walkstations. This is achieved by adopting either the disconnected or the distributed approaches.

Applications exist whose transactions intrinsically have a **long** duration. Moreover, in the mobile setting transactions are likely to have longer duration than in the fixed setting, because of the intrinsic asynchronicity of the system. Under these conditions the distributed approach can be unsuitable. In fact, it forces each core site to stay connected throughout the whole transaction execution, even if the site has not initiated a transaction. The length of the connection time makes the distributed approach costly in terms of power consumption. The disconnected and the transaction proxy approaches are better suited than the distributed approach in the case of long transactions.

The described approaches differ in their capability of supporting **conversational** (interactive) transactions. Conversational transactions are not possible in the transaction proxy approach if MHs are forced to submit whole transactions to MSSs [13]. On the contrary, if transactions are split into subtransactions, interactive transactions can be submitted. Moreover, in this case, the requirement that subtransactions of the same transaction must be independent can be relaxed. In fact, consider a subtransaction $\mathcal{S}_i$ of a conversational transaction $\mathcal{T}$, initiated by a user on MH_m. Let $\mathcal{S}_i$ be generated as a result of an interaction of the user with the mobile application. Then, $\mathcal{S}_i$ is necessarily processed after the MH_m has received the results of the previously submitted subtransactions. Hence, no concurrent processing is possible between $\mathcal{S}_i$ and the other subtransactions of $\mathcal{T}$ on which $\mathcal{S}_i$ depends. No inconsistency can arise even if $\mathcal{S}_i$ is not independent from the previous subtransactions.

The ability of exploiting **concurrency** amongst transactions is interesting for both the increase in the throughput and the possibility the MHs have of working autonomously from each other, possibly in disconnected mode. In the case transaction processing is carried out by FHs, such as in the transaction proxy approach, the usual concurrency control policies are used. In the distributed approach, the degree of the achieved concurrency depends on the locking policy. For instance, in the read-one-write-all approach all the queries submitted at different core MHs can be executed concurrently. This does not occur in the majority quorum approach. The disconnected approach, in spite of the high degree of achievable concurrency among disconnected MHs, is heavily affected by conflicts amongst transactions. In fact, as pointed out in [28, 30], this approach is not adequate for databases in which a fine granularity level is used for locks and highly concurrent applications are executed.

A better load balancing of transactions can be achieved in **multiversion** databases, if applications can tolerate data not up-to-date.

Required underlying services

To fully evaluate each approach, it is important to consider which services each approach requires to the underlying protocol layers and which assumptions it makes on the characteristics of the underlying system. All of the approaches that use the wireless communication within a single cell do not require any underlying service.

The distributed approach described in [7] requires some form of **group addressing** and **group oriented services**. The multicast protocol proposed in [1, 2] could be used to this purpose. Such protocol guarantees an exactly-once, reliable and ordered message delivery service to a group of mobile hosts. Furthermore, if the quorum or the read-one-write-all methods are chosen, an **agreement** protocol is required. To avoid system deadlocks, the core copies must agree on the order at which they have to satisfy the lock requests.

A **location service** [5, 25] is also required. As long as the hosts that store core copies do not disconnect, the copies can be moved across cell boundaries. Hence, the initiator of a transaction on a host MH_m, that must lock other core copies, has to know their current location. This service is generally provided at the network layer.

Protocols using timestamps [21, 44] to check the validity of data, require some form of **synchronization** amongst all the hosts. This property is, however, quite difficult to achieve in a mobile environment.

The above mentioned requirements restrict the applicability of the described approaches to those systems which provide the required services. An interesting question is whether and to which extent these protocols affect the performance of the upper layer algorithms for database management. As far as we know no investigations have been reported on this issue.

Power saving

An important comparison aspect concerns the ability of carefully managing the available power resources. In the remainder we list the characteristics of the approaches that allow to save energy, together with some considerations about how they are fulfilled by the different approaches.

Use of the wireless network. As pointed out in section 1.2, the use of the hardware interface to the wireless network implies high power consumption. Hence, MHs should stay disconnected as long as possible.

Under this point of view, the most expensive approach is the distributed approach, in which communications can span over different cells.

On the contrary, in both the transaction proxy approach and the disconnected approach communications take place *inside* the cell of the MH_m that submitted a transaction. However, in the last case, the size of the exchanged messages can be larger, because they contain either the data, that are needed during the disconnection, or the log file that reports the activities being carried out by weak transactions and still globally uncommitted.

The use of server callbacks [21, 28, 30] to invalidate MH caches optimizes the use of the wireless network because, as long as the cache remains valid, no communication is required. The relocation of the computation as near as possible to the current location of the submitting MH_m has a similar effect, because wireless communication is confined inside a cell.

Possibility of supporting disconnections. This issue involves two requirements: (*i*) the possibility of working in disconnected mode, thus reducing power consumption, and (*ii*) the possibility of staying doze or disconnected every time the MH_m is not directly involved in transaction processing. These requirements are not satisfied by the distributed approach, as we have seen above.

Computation load on MHs. Although walkstations have computing capabilities comparable to those of fixed workstations, the use of these computing resources implies power consumption. According to the requirements of the applications regarding MHs autonomy, it is preferable to maintain the computation on FHs as longer as possible. This goal is achieved by the data-broadcasting and the transaction proxy approaches. Moreover, if the MHs are actively involved in the computation, only the MHs directly concerned with the transactions (i.e. their initiators) should process the transaction. This guideline has not been met in the distributed approach. Moreover, the choice of the lock policy in the distributed approach affects the distribution of the computation load. With the primary/backup method the entire computation is carried out by the current primary site, whose power resources are heavily used. By contrast, with the quorum methods the load of submitted transactions can be balanced among different subsets of hosts.

1.7.2 Fault-tolerance and recovery issues

In a mobile environment, where the MHs failures are indistinguishable from disconnections, the fault-tolerance is guaranteed by re-defining the correctness notion, namely the ACID properties, to properly meet different application requirements. In the resulting fault-tolerant architecture, the achievement of the needed level of consistency, durability and availability over replicated data is solved at the application level according to the well known end-to-end argument [37]. All the described approaches follow this schema that seems to be more flexible and efficient than the construction of a middleware layer that provides some general purpose data management services. The transport protocol is still responsible of detecting and recovering failures over wireless channels.

1.8 PERFORMANCE EVALUATION OF RECOVERY TECHNIQUES

The data management techniques introduced in section 1.6 cannot be meaningfully compared with respect to performance, because they differ both in the system assumptions and in the application requirements they address. However, it appeared of some interest to compare the four techniques that apply to the distributed approach, because they directly derive from wired systems and because they require tightly coupled and connected operations. As a consequence, these techniques are prone to be somehow affected by mobility. In line with that, we used simulations to observe the following indexes:

throughput: the number of transactions that commit in the system per time unit. It depends on both the specific transaction management policy and the concurrency control method. It is affected by (*i*) the amount of communication on the wireless network and the size of the exchanged messages; (*ii*) the amount of computation on the MHs; (*iii*) the need of locating MHs within the system as a consequence of mobility;

latency: it can be observed at the interface with the application processes and it is computed as the elapsed time between the transaction generation time and the commit time. It depends on the same factors that influence the throughput. Moreover, the latency can be negatively affected by the possibility of conflicts among transactions. The conflict probability depends on both the concurrency degree amongst users and the rate between read and write operations.

scalability: the capability of scaling is fundamental in the mobile environment, where the number of hosts involved in a distributed protocol is expected to be higher than in the fixed setting. It is easy to notice that the most scalable among the described approaches is the data-broadcasting approach. Indeed, only one broadcast message is required inside a cell m to satisfy all the waiting queries, independently from the number of listening MHs within m.

Our measures concern:
(i) the primary/backup method with the primary site residing on a FH (P_FH);
(ii) the primary/backup method with the primary site residing on a MH (P_MH);
(iii) the read-one-write-all method (R.o.w.a.);
(iv) the majority quorum method (Maj.).
Cases (i) and (ii) allow to evaluate the impact of the use of the wireless network. Case (i) strictly resembles the transaction proxy approach in which MHs must submit whole transactions to their MSSs.

The main difficulty derives from the determination of the appropriate parameters that characterize the mobile environment. Examples of these parameters are the number of MHs that can simultaneously be involved in the same distributed application, and the concurrency level amongst MHs in accessing the database (e.g. in terms of the rate between queries and updates). The difficulty in estimating the parameters is a consequence of the lack of experience on real mobile systems and applications. We believe, however, that these parameters do not much differ from those in the fixed setting.

The system we considered is composed of 5 MSSs and 200 MHs amongst which core copies are uniformily distributed. The ratio we used between the bandwidth of the wired and the wireless links is equal to $1:3$. In particular, we simulated a wireless network having a bandwidth of 19.2 Kbps and a wired network having a bandwidth of 64 Kbps; clearly, results would change if faster existing technologies were considered. Core copies can move from one site to another. We chose appropriate parameters to model a system in which core sites change 3-4 times a minute, much higher than in real systems. We did not model sudden disconnections, because the distributed approach does not tolerate them. The load in terms of transactions is decided by comparing the value of a random variable, generated according to a β distribution, with 0.71.

In Figure 1.9(a), we compare the throughput produced by the various methods under different load conditions. On the x-axis we report the mean of the β distribution. A group of 5 core sites has been simulated, whose access pattern to the database consists of a 70% of read operations (queries) and a 30% of write operations (update transactions).

The P_FH approach takes great advantage of its reduced use of the wireless network with respect to all the other approaches. In fact, messages are never exchanged between a pair of MHs. The P_MH approach seems to suffer from the growing load on the wireless link. This is a consequence of the multicast protocol we used. This protocol guarantees a reliable, ordered message delivery. Under this protocol, a MSS uses blocking primitives to send the messages to the MHs in its cell. For this reason, in the event of write operations the primary MH wastes more time than the primary FH in receiving the acknowledgements to the backup messages,

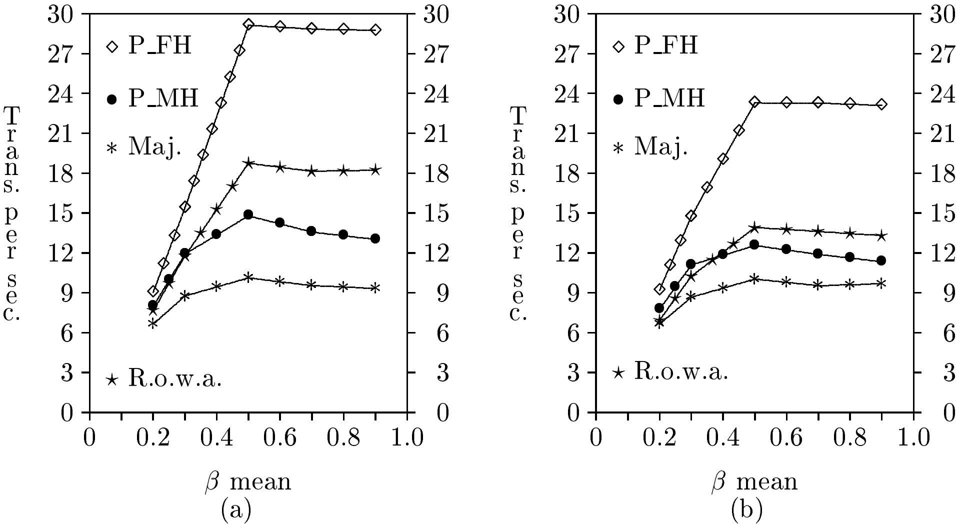

Figure 1.9. Throughput of the lock methods vs. load under different access patterns: (a) 70% read operations and 30% write operations; (b) 50% read operations and 50% write operations

from the other core sites. The overhead grows with the increase in the transaction load thus leading to system congestion.

Both the distributed methods generate the highest number of messages that traverse wireless links. However, the majority quorum method is further penalized because it does not exploit concurrency amongst read operations.

These considerations are confirmed by the results shown in Figure 1.9(b), where the rate of read and write operations has been respectively set to 50% and 50%. The throughput is lower for almost all the approaches because of the exchange of a higher amount of backup messages generated as consequence of write operations. The read-one-write-all method is more sensitive to the higher amount of write operations because MHs are not able to fully exploit concurrency amongst read operations. The majority approach shows the same behaviour in the two cases because backup messages are sent together with the unlock messages. As a consequence, the number of messages is independent from the access pattern.

In the experiments reported in Figure 1.10(a) we introduce a 10% of failure probability on each wireless link, while the wired network is still reliable. We report the observed throughput as the percentage of the throughput obtained in the reliable case, under the same conditions of Figure 1.9(a). As a consequence of failures, the throughput decreases according to the increase in transaction load. The read-one-write-all method results to be less sensitive to failures than the other approaches because it generates the lowest amount of messages. On the contrary the P_FH approach shows the worst performance.

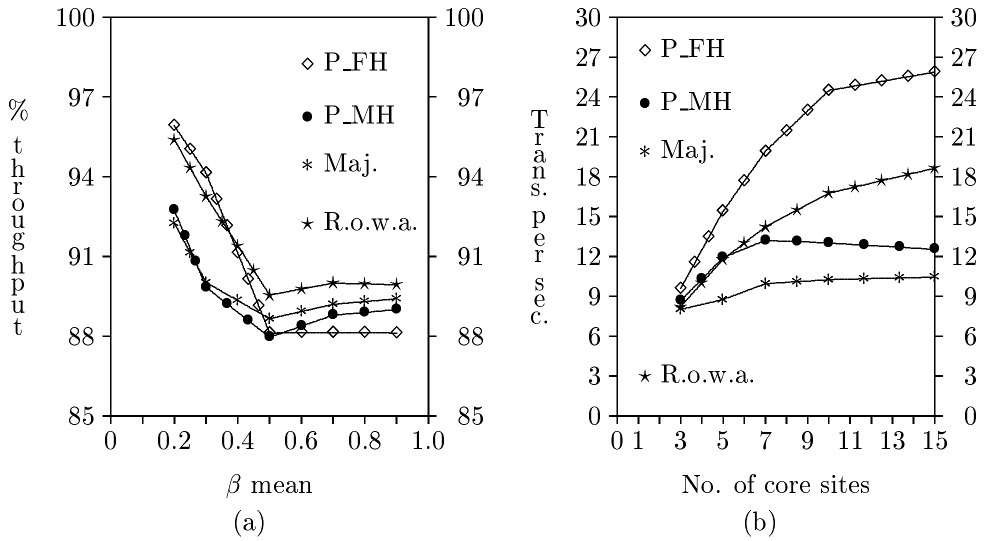

Figure 1.10. (a) Percentage of throughput in presence of 10% link failures. (b) Throughput vs. number of core sites as a measure of scalability.

To evaluate the scalability we performed the same measures of Figure 1.9(a) varying the number of core sites at a fixed β mean (0.3). Results are reported in Figure 1.10(b). The increase in the number of core sites has the same effect of the increase in the load. The methods show a comparable behaviour as in Figure 1.9(a).

In Figure 1.11 we report the time spent by the different methods to complete a query (Figure 1.11(a)) or an update (Figure 1.11(b)). Measures have been taken under the same conditions of Figure 1.10(b). The observed performance clearly differentiates the P_MH and majority quorum approaches from the P_FH and read-one-write-all approaches. The P_MH and the majority quorum approaches are affected by the high number of exchanged messages. Figure 1.11 confirms that the P_MH method is also affected by the congestion on the wireless links when backup messages are generated as a consequence of write operations. By contrast, the increase in the number of generated backup messages has a lower impact on the latency in the case of the P_FH method, because of the reduced use of the wireless network. As we could expect, the read-one-write-all method has a very different behaviour when executing either read or write operations. The queries submitted at different hosts can be all processed concurrently. Hence, the measured time only accounts for the concurrency with write operations and thus the delay in acquiring the grant to lock the local copy of the database.

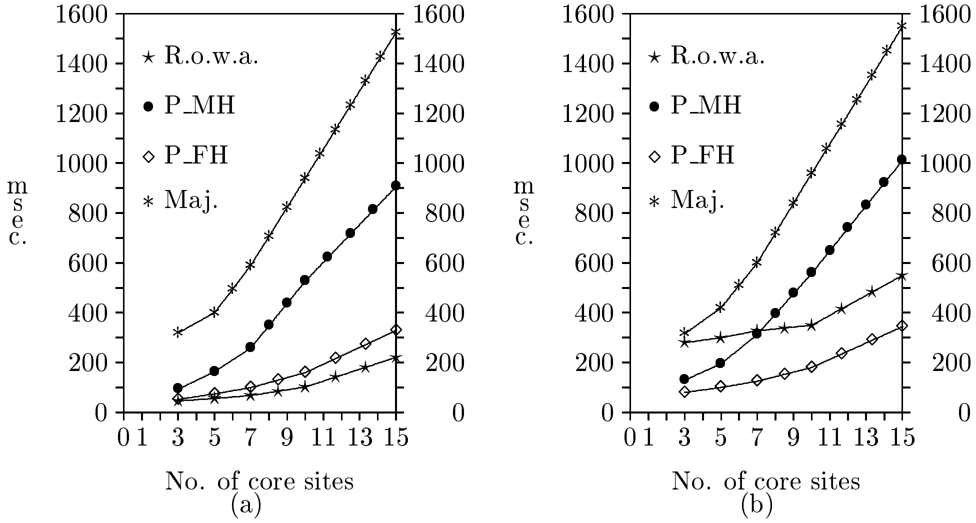

Figure 1.11. Comparison amongst lock methods with respect to the latency for (a) read and (b) write operations

1.9 CONCLUSIONS

In this chapter we have investigated database management issues in a mobile computing environment. We have described the characteristics of the mobile setting and we have illustrated how these characteristics impact the algorithms for data management. We have discussed and compared some of the approaches proposed for guaranteeing proper data maintenance in spite of mobility and disconnections. We have analyzed the performance of some of the described approaches.

The simulation results highlight that the effect that the presence of the wireless network has on the algorithms is the increase of the latency. The development of new technologies will enable faster communications, while the reliability of the wireless links can be improved by using appropriate transport protocols, thus reducing the latency. Also mobility seems to have no other effects than the introduction of delays.

The impact of disconnections depends on the approach adopted for data management. In the disconnected, the transaction-proxy and the data-broadcasting approaches disconnections are supported to improve efficiency in power saving. By contrast, the distributed approach does not tolerate sudden disconnections.

In turn, the choice of a specific approach depends on the characteristics of the application. For instance, the distributed approach is probably the most suited for managing data accessed by highly concurrent users, e.g. in cooperative workgroup applications, in spite of its cost. If, however, the probability that the same data be simultaneously accessed by more than one user is low, the disconnected approach should be preferred. When conflicts are rare, the disconnected approach allows a better resource usage than the distributed approach. The choice of an approach is also driven by the characteristics of the system. When MHs are dumb terminals, only the data-broadcasting and the transaction proxy approaches are possible.

Several research issues remain open. Extensive performance measurements should be carried out to further analyze and compare the existing approaches. In particular, it would be interesting to estimate the time a MH has to stay connected for transaction processing according to the different approaches. These evaluations would allow to determine a trade-off between the possibility of disconnected work and the probability that a weak transaction has to be re-executed after having globally aborted because of conflicts.

Protocols could be designed to add fault-tolerance to the algorithms that implement the distributed approach, so that sudden disconnections are tolerated. Such protocols should probably run underneath the data management algorithms.

The distributed approach deserves much attention, because it is the most suitable for multi-hop networks. Since this approach makes no distinction between MHs and FHs, nor relies on some characteristics specific to FHs for its correctness, all the involved hosts could be MHs. On the contrary, the other approaches could be used in multi-hop networks only by assuming the existence of particularly reliable and powerful MHs that could act as FHs with respect to the other hosts in the system.

References

[1] Acharya A., Badrinath B. R. "A Framework for Delivering Multicast Messages in Networks with Mobile Hosts", ACM/Baltzer Journal of Wireless Networks, Special Issue on *Routing in Mobile Communication Networks*, Vol. 1, 1996. ftp://athos.rutgers.edu/pub/acharya/mcast_winet.ps.Z.

[2] Acharya A., Badrinath B. R. "Delivering Multicast Messages in Networks with Mobile Hosts", *Proc. 13th Intl. Conf. on Distributed Computing Systems*, May 1993.

[3] Badrinath B. R., Acharya A., Imielinski T. "Designing Distributed Algorithms for Mobile Computing Networks", To appear in Computer and Communications, 1996. ftp://athos.rutgers.edu/pub/acharya/algos_journal.ps.Z.

[4] Badrinath B. R., Acharya A., Imielinski T. "Impact of Mobility on Distributed Computations", *ACM Operating Systems Review*, Vol. 27, No. 2, April 1993, pages: 15–20.

[5] Badrinath B. R., Imielinski T. "Replication and Mobility", *Proc. 2nd IEEE Workshop on Management of Replicated Data*, November 1992.

[6] Balakrishnan H., Seshan S., Katz R. H. "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks", *ACM Wireless Networks*, 1996.

[7] Barbará D., Garcia-Molina H. "Replicated Data Management in Mobile Environments: Anything New Under the Sun ?", *Proc. IFIP Conf. on Applications in Parallel and Distributed Computing*, Caracas, Venezuela, Apr. 1994.

[8] Bassak G. "Brainy, Brawny Batteries", *Byte*. Special Issue on "Mobile Computing", June 1995.

[9] Bernstein A. J. "Program Analysis for Parallel Processing", *IEEE Transactions on Electronic Computers*, Vol. EC-15, No. 5, Oct. 1966.

[10] Chen M., Yu P. S., Wu K. "Index Allocation for Sequential Data Broadcasting in Wireless Mobile Computing", *IBM T. J. Watson Research Center Technical Report*, 1994.

[11] Chlamtac I., Weinstein O. "Distributed 'Wave' Broadcasting in Mobile Multi-Hop Radio Networks", *Proc. 7th Intl. Conf. on Distributed Computing Systems*, 1987.

[12] Chlamtac I., Weinstein O. "The Wave Expansion Approach to Broadcasting in Multihop Radio Networks", *IEEE Transactions on Communications*, Vol. 39, No. 3, Mar. 1991.

[13] Cho K., Birman K. P. "A Group Communication Approach for Mobile Computing", *Proc. Mobile Computing Workshop*, Santa Cruz, CA, Dec. 1994.

[14] Coulouris G., Dollimore J., Kindberg T. "Distributed Systems: Concepts and Design - 2nd Edition" Addison-Wesley, 1994.

[15] Duchamp D., Tait C. D. "An Interface to Support Lazy Replicated File Service", *Proc. 2nd IEEE Workshop on Management of Replicated Data*, Monterey, CA, Nov. 1992.

[16] Elmagarmid A. K. "Database Transaction Models for Advanced Applications", Morgan Kaufman.

[17] Eswaran K. P., Gray J. N., Lorie R. A., Traiger I. L. "The Notions of Consistency and Predicate Locks in a Database System", *Communications of the ACM*, Vol. 19, No. 11, 1976.

[18] Garcia-Molina H., Salem K. "Sagas", *Proc. ACM Conf. on Management of Data*, May 1987.

[19] Gray J., Reuter A. "Transaction Processing: Concepts and Techniques", Morgan Kaufmann, 1993.

[20] Haerder T., Reuter A. "Principles of Transaction-Oriented Database Recovery", *ACM Computing Surveys*, Vol. 15, No. 4, Dec. 1983.

[21] Helal A., Balakrishnan S., Dunham M., Elmasri R. "A Survey of Mobile Transaction Models", *Technical Report* CSD-TR-96-003, Department of Computer Science, Purdue University, Feb. 1996.

[22] Huang Y., Wolfson O. "Object Allocation in Distributed Databases and Mobile Computers", *Proc. 10th Intl. Conf. on Data Engineering*, Houston, TX, Feb. 1994.

[23] Imielinski T., Badrinath B. R. "Mobile Wireless Computing: Challenges in Data Management", *Communications of the ACM*, Vo. 37, No. 10, 1994.

[24] Imielinski T., Badrinath B. R. "Mobile Wireless Computing: Solutions and Challenges in Data Management", Rutgers University Tech. Rep. DCS-TR-296/WINLAB-TR-49, 1994.

[25] Imielinski T., Badrinath B. R. "Querying Locations in Wireless Environments", Wireless Communications: Future Directions, J. M. Holtzmann, D. J. Goodman. Kluwer Academic Publishers, 1993.

[26] Imielinski T., Viswanathan S., Badrinath B. R. "Energy Efficient Indexing on Air", *ACM SIGMOD*, May 1994.

[27] Jing J., Bukhres O., Elmagarmid A. "Distributed Lock Management for Mobile Transactions", *Proc. 15th Intl. Conf. on Distributed Computing Systems*, Vancouver, British Columbia, Canada, Jun. 1995.

[28] Kistler J. J., Satyanarayanan M. "Disconnected Operation in the Coda File System", *ACM Trans. on Computer Systems*, Vol. 10, No. 1, Feb. 1992.

[29] Liu G., Marlevi A., Maguire G. Q. "A Mobile Virtual-distributed System Architecture for Supporting Wireless Mobile Computing and Communications", *Proc. ACM MOBICOM '95*, Nov. 1995.

[30] Lu Q., Satyanarayanan M. "Isolation-Only Transactions for Mobile Computing", *ACM Operating Systems Review*, Vol. 28, No. 2, Apr. 1994.

[31] Perkins C. "IP Mobility Support Draft 12. IETF Mobile-IP Draft", 1995,

[32] Pitoura E., Bhargava B. "Revising Transaction Concepts for Mobile Computing", *Proc. 1st IEEE Workshop on Mobile Systems and Applications*, Santa Cruz, CA, Dec. 1994.

[33] Pitoura E., Bhargava B. "Maintaining Consistency of Data in Mobile Distributed Environments", *Proc. 15th Intl. Conf. on Distributed Computing Systems*, Vancouver, British Columbia, Canada, Jun. 1995.

[34] Pu C., Kaiser G., Hutchinson N. "Split-transactions for Open-ended Activities", *Proc. 14th VLDB Conference*, 1988.

[35] Reinhardt A. "From Here to Mobility", *Byte, Special Issue on "Mobile Computing"*, Jun. 1995.

[36] Salamone S. "Radio Days", *Byte. Special Issue on "Mobile Computing"*, Jun. 1995.

[37] Saltzer J. H., Reed D. P., Clark D. D. "End-to-End Arguments in System Design", *ACM Transactions on Computer Systems*, Vol. 2, No. 4, Nov. 1984.

[38] Seshan S., Balakrishnan H., Katz R. H. "Handoffs in Cellular Wireless Networks: The Daedalus Implementation and Experience", *Technical Report*, Dept. of Electrical Engineering and Computer Science, Univ. of California at Berkeley, 1995.

[39] Tait C. D., Duchamp D. "An Efficient Variable-Consistency Replicated File Service", *Proc. USENIX File Systems Workshop*, Ann Arbor, MI, May 1992.

[40] Tait C. D., Duchamp D. "Service Interface and Replica Management Algorithm for Mobile File System Clients", *Proc. 1st IEEE Intl. Conf. on Parallel and Distributed Information Systems*, Miami, FL, Dec. 1991.

[41] Wirth P. E. "Teletraffic Implications of Database Architectures in Mobile and Personal Communications", *Computer Networks and ISDN Systems*, Vol. 28, No. 5, Mar. 1996.

[42] Wong M. H., Leung W. M. "A Caching Policy to Support Read-Only Transactions in a Mobile Computing Environment", *Technical Report*, Dept. of Computer Science, Chinese Univ. of Hong Kong, 1994.

[43] Wu K., Yu P. S., Chen M. "Energy-Efficient Caching for Wireless Mobile Computing", *IBM T. J. Watson Research Center Technical Report*, 1994.

[44] Yeo L. H., Zaslavsky A. "Submission of Transactions from Mobile Workstations in a Cooperative Multidatabase Processing Environment", *Proc. 14th Intl. Conf. on Distributed Computing Systems*, Poznan, Poland, Jun. 1994.