In this chapter, the issues concerned with mobile database transactions are identified through a detailed literature review. Traditional transaction models are investigated to find out their suitability for applying them to mobile computing environment. Existing mobile transaction models are also investigated and their strengths and weaknesses are observed. Then the concurrency control mechanisms in mobile database system and commitment of mobile transactions are reviewed. Finally the invalidation report caching schemes in mobile environment are reviewed.

4.1 Traditional Transaction Models

The discussion on transaction management begins with a flat transaction leading to more complex form of advanced transaction models. The traditional transaction models make sure that ACID properties of transaction are maintained during database processing [8]. A number of traditional transaction models are reviewed in this section: (a) Flat transaction (b) Nested transaction (c) SAGA (d) Cooperative transaction (e) ConTract and (f) Flex transaction.

4.1.1 Flat Transaction Model

A flat transaction [54, 8] is the simplest type of mechanism which supports one-level operation; that is, during their execution they do not trigger other dependent transactions. The flat transaction model fully meets the standard ACID properties. The flat transaction is fully isolated during its execution, and any failure causes the whole transaction to abort. The results of a committed flat transaction are durable and
permanent. Due to the strict ACID properties, the flat transaction model is not suitable to mobile environments. However, the flat transaction model plays an important role for building more advanced transaction models. Some examples of flat transactions are debit/credit transaction, bill payment transaction, customer account statement transaction and so on.

4.1.2 Nested Transaction Model

Nested transaction model [55] is organized as a tree. The root of the tree represents the parent transaction and descendants of the root represent finer granularity subtransactions of the parent transaction. Subtransactions at the leaf level of the transaction tree are flat transactions. The relationship between the root and its descendants are clearly defined and are used to manage consistency preservation execution (commit, rollback, abort, etc.) of the entire transaction tree. The relationship is summarized as given below.

- Flat transactions at the leaf level actually perform the necessary data processing. The parent of a set of subtransactions does not manipulate any data but invokes subtransactions and controls the flow of execution.
- A subtransaction can commit or rollback independent of other subtransactions or the parent of the tree. However, such a commitment or rollback takes effect only if the parent commits or rolls back.
- A subtransaction has only A, C, I properties because its durability depends only on its parent. If the parent rolls back, then all its subtransactions must also be rolled back.

Concurrent parent transactions conform to ACID properties; therefore, their execution must be serialized through some serialization scheme. The parent
transaction has a number of options in dealing with the failure of a transaction: (a) A parent can ignore the failure of subtransactions if it does not threaten database consistency. (b) It can execute a special subtransaction (contingency subtransaction) to make up the failure of the subtransaction. (c) It can reexecute the failed subtransaction. (d) It can abort an active subtransaction. (e) It can reschedule the execution of set of subtransactions.

4.1.3 SAGA Transaction Model

The saga transaction model [56] was developed using compensating transaction [57] and long running transaction [8]. In this model, a long running transaction is divided into a number of independent transactions that are chained together so that independent execution of each transaction is possible. The compensating transaction will compensate if there is any malfunctioning (rollback, abort, incorrect operation, etc.) of a single transaction without affecting others.

The SAGA model can be formally defined as a set of fine granule transactions (subtransactions) (T) where each subtransaction is associated with a compensating transaction (C). The SAGA can be viewed as a unit of execution with two sequences; (a) a sequence of subtransactions (b) sequence of compensating transactions.

\[ T_1, T_2, \ldots, T_n \]

\[ C_1, C_2, \ldots, C_n \]

The system either executes the first sequence successfully or in the case of failure, it runs sequence (a) and a part of sequence (b). During the execution of a sequence (a), any subtransaction can commit or rollback independently; that is, it does not wait for other transactions of the same saga. In case of failure, if forward recovery
is not possible, then it uses the corresponding compensating transaction to remedy the situation. In this remedy, the compensating transaction may correct the error of the subtransaction or may undo all the operations on the database.

This kind of execution environment is achieved by saga by relaxing some of the ACID properties as follows.

- **Atomicity:** Unlike closed nested transaction, saga allows other sagas to see the partial results of a transaction. As a result, saga cannot guarantee complete atomicity.
- **Consistency:** Unlike flat transaction, saga allows interleaved execution of subtransactions where an active subtransaction can see the partial results of other active subtransactions. Under this execution consistency may not be guaranteed.
- **Isolation:** Since consistency is relaxed isolation cannot be guaranteed either.
- **Durability:** This is preserved because saga guarantees that the main transaction commits only when all its subtransactions are committed.

### 4.1.4 Co-operative Transaction

In [58], the cooperative transaction is modelled as a rooted tree. The internal nodes and the leaves are known as transaction groups and cooperative transactions respectively. A number of cooperative transactions are the children of a transaction group which is a parent. The activities of the cooperative transactions are monitored by a transaction group. The interactions between the siblings are managed by external protocols.
Every transaction group has its own version of a data item which can be accessed only by its cooperating members. A transaction group itself should be a cooperative transaction in order to take part in data processing activities. When there is a request for a data item by a member, then it is automatically copied from the group to the requesting member. In this way, a transaction group acts as a server for its cooperative transactions.

The cooperative transaction members of a transaction group operate under internal protocol. They act like clients that request its transaction group to perform read and write operations. The transaction group in turn, can process, refuse, or queue the request for later processing. A transaction group monitors the cooperation among members for serialization. For the members, execution flow starts with begin transaction, end transaction and commit. From an execution view point, the flow is similar to the execution of a flat transaction.

4.1.5 ConTract Transaction Model

This model [59] proposes a scheme for developing, implementing and managing long-lived transaction. Using this approach, a fault tolerant execution system can be built to manage the execution of an arbitrary sequence of predefined actions, referred to as steps, according to a script. ConTract is a program which contains well-defined error semantics for managing system recovery. It also provides a parallel processing environment.

In ConTract, step is similar to a subtransaction which is executed sequentially. Though a step is implemented as an ACID transaction, it takes care of only local consistency of the data items. Using script, the flow of control can be defined by means of loop, iteration, recursion, etc.
The ConTract model is different from the ACID model in the following ways.

**Atomicity:** Interruption in the execution of a ConTract step can occur at any time so that its execution can be frozen. Later on, the execution can be restarted from the point of interruption. When failure takes place, for the purpose of maintaining atomicity, only roll forward of a step, which may use a different execution path than the original one is allowed.

**Consistency:** System integrity is maintained by a ConTract on a large scale.

**Isolation:** A ConTract uses semantic isolation instead of locking to achieve a high degree of concurrency.

**Durability:** Conventionally, A ConTract execution is durable. However, only by running another ConTract, a committed ConTract can be compensated.

### 4.1.6 Flex Transaction Model

The flex transaction model proposed in [60] is an extension of flat transaction model. This model is developed for the purpose of managing transactions on multi and federated database systems. A flex transaction consists of a set of tasks and each task contains a set of functionally equivalent subtransaction. The main aim of this model is to minimize ACID constraints in order to have more flexibility in processing concurrent transactions. This is achieved by giving more freedom to users so that they can define their data processing preferences. Relaxation in the commitment requirement in this model allows a flex transaction to commit even if some of its task did not complete.

In this model, function replication is possible to allow enough flexibility in composing and processing global transactions. A flex transaction is known as a mixed...
transaction since it consists of compensatable as well as non-compensatable subtransactions.

4.2 Mobile Transaction Models

The introduction of mobility significantly changed the database architecture and management paradigm, and it became clear that strict enforcement of ACID properties was not necessary to maintain database consistency. The mobility changed and in many cases had to relax the notion of consistency because in mobile database systems the notion of consistency is closely related to locations in the geographical domain.

The features of traditional transaction models are still useful in mobile computing environment. But these traditional models are not suitable to meet the challenges of mobile transactions such as mobility and disconnection issues. In order to meet these challenges, many advanced transaction models have been developed to specially support mobile transactions. In this section, the following mobile transaction models are reviewed.

4.2.1 Reporting Transaction Model

Reporting [61] is based on open nested transaction model [55, 62, 63]. Reporting considers transactions on MHs as a set of subtransactions in a mobile database system. The parent ACID transaction is considered as a set of component transactions. Each component transaction is divided into a number of low level component transactions. During the execution, partial results can be shared by the transactions and the state of a mobile subtransaction is partially maintained on a MSS. There can be four types of component transactions: (a) Atomic transaction can be
structured with begin, commit and abort properties. (b) A non-compensatable
transaction which is a component of a parent transaction can delegate to its parent all
updates at the time of commit (c) reporting transaction report its updates to another
transaction at any time during its execution. (d) Co-transaction is a reporting
transaction whose execution is suspended when it reports to the parent transaction and
resumes its execution from the point of suspension.

This model also proposed vital component and non-vital components. The
parent commits only after all the vital components committed. For non-vital
components, there is no requirement of this commit dependency. A mobile transaction
is considered as a set of transactions, of which reporting components always execute
on base station and co-transactions can be executed on the MHs.

4.2.2 Two Level Consistency Model

Two level consistency model was proposed in [64, 65], where the data base is
divided into clusters of semantically related or closely located data which were
dynamically configured. It offers a replication approach to mobile computing
environment to overcome the problem of disconnection variation. A cluster is
distributed over several strongly connected hosts. An MH becomes a cluster once it is
disconnected.

Two copies exist for every object: the strict version and the weak version.
Strict version should be globally consistent while the weak version could tolerate
some degree of global inconsistency but must be locally consistent. In this model
mobile transactions can be either strict or weak. Weak and Strict transactions access
weak and strict versions, respectively. It also introduces weak read and weak write.
Strict transactions consist of standard reads and writes while weak transactions consist
of weak reads and weak writes. The weak copy is used when mobile hosts are disconnected or connected via a slow and unreliable network. During reconnection, a synchronization process reconciles the changes of the local data version with the global data version which makes the database globally consistent.

4.2.3 Pro-motion

In [66, 22], a model called Pro-motion (Proactive management of mobile Transactions) was proposed. Pro-motion can be considered as a mobile transaction processing system which supports disconnected transaction processing in a mobile environment. Pro-motion takes into account the data caching problem on MHs so that local transaction processing is done in a consistent mode. Compact is used as the basic unit of replication for caching, prefetching and hoarding. In the construction of compacts, in order to improve concurrency, object semantics is exploited. This model makes use of nested split transactions [61, 68].

In Pro-motion, the entire mobile sub-system is considered as one extremely large, long-lived transaction which executes at the server with a subtransaction executing at each MH. Each of these MH subtransactions, in turn, becomes the root of another nested split transaction. Compact manager at the server, handles the creation and removal of compact. The management of compacts is the responsibility of the compact manager, a compact agent at the MH and the mobility manager at the Base Station. Compact manager stores the status of each compact. Whenever the status of a compact changes, the state of the compact is updated at predefined intervals. The compact manager acts as a front end for the database server and shields it from mobile hosts where transaction execution takes place. Database server can interact with a compact manager as an ordinary database client executing a single large long lived
transaction. In an MH, compact agent is responsible for concurrency control, logging and recovery. Transmissions between agents are taken care of by mobility manager. When the mobile host is reconnected to the server, compact agent will reconcile local updates with the server. This synchronization process will check compacts which have been modified by the locally committed transactions. Global commit will be performed, if compacts are able to maintain global consistency.

4.2.4 Pre-write Transaction Model

In [69, 70], Pre-write transaction model that increases data availability in mobile environments was proposed. Mobile transactions can be initiated at the mobile host. In this model, the increase in data availability is achieved by allowing a transaction on a mobile host to commit on logical level and make its data available to other concurrent transactions. The logical commit is achieved by means of pre-write operations. Then it issues a pre-commit state to the mobile support station. After that, the rest of the mobile transaction can be carried out. Finally commitment at fixed hosts can take place, only when it satisfies certain predefined constraints. Between the pre-committed result and the final committed result a small variation is allowed. Pre-committed data values are accessible to other transactions via pre-read operations. Two different types of lock, which are the pre-read and pre-write are introduced to support the new operations. Mobile transactions are not allowed to abort after they have submitted pre-commit operations to the mobile support station. This mobile transaction model can be used to support mobile hosts which have little or no capacity for transaction processing.
4.2.5 Semantic Based Model

A semantic-based transaction processing method has been extended for mobile transactions in [85]. This model introduces the idea of fragmentable and reorderable objects to increase the concurrency to the entire database and improve the efficiency of caching. Using this idea, a mobile system may request a locally cached fragment on which transactions may operate. This fragment is consistent in that it is only accessible to the transactions of the mobile computer that requested it. Once the fragments are no longer required, they are merged back into the master database. Such a technique is suitable when data objects may be fragmented like aggregate items, sets and stacks. Mobile transactions are long-lived because of communication delays from the database server point of view. MH fragment request includes two parameters: selection criteria and consistency conditions. The selection criteria indicate data to be cached on the MH in the required fragment size. The consistency conditions specify constraints to preserve consistency on the entire data.

4.2.6 HiCoMo (High Commit Mobile Transaction Model)

This model [71] is used for the purpose of processing aggregate data in a data warehouse which resides in mobile hosts. HiCoMo transactions are initiated on the mobile hosts in a disconnected mode. Therefore, commit operations are done very fast. At reconnection, the updates are reconciled with the server.

Base or source transactions manipulate the base database that resides on the fixed network. HiCoMo transactions are initiated and executed at MHs. The data warehouse consists of aggregate data that represent summary or statistics (e.g. average, summation, minimum, maximum). The following operations are used with
some conditions on their order of application: addition, subtraction and multiplication. Some margin of errors is allowed by this model.

The HiCoMo transaction is derived from the nested transaction model. Convergence criteria are used to satisfy database consistency. Convergence occurs when the state of the base database becomes identical with the state of the data warehouse in the MHs. HiCoMo model ensures that convergence is always satisfied. Base transactions reflect the modifications made by HiCoMo transactions on base tables. Thus to install updates of HiCoMo transactions, they must be transformed into base transactions. This transformation is done as given below using conflict detection and base transaction generation.

Conflict detection is carried out to identify if there is a conflict among HiCoMo transactions and between HiCoMo and base transactions. In case there is a conflict among HiCoMo transactions, then the transaction which is considered for transformation is aborted. If there is no conflict, base transactions are generated and executed as subtransactions at the server.

4.2.7 Moflex Transaction Model

The Moflex transaction model presented in [72] is built on top of multi-database systems and based on the concepts of split-join transactions. It is an extension of the Flexible transaction model [60] developed for heterogeneous mobile database system where a transaction consists of subtransactions which are related by a set of execution dependencies such as success, failure and external dependencies (time, cost, or location). Apart from flexible transactions, this model takes into account location dependent subtransactions [73]. It also supports subtransaction execution when hand off takes place.
A Moflex transaction is initiated by the mobile host. A Moflex transaction consists of compensable or non-compensable subtransactions. The subtransactions are submitted to the mobile transaction manager that resides at the mobile support station. The mobile transaction manager will send these sub-transactions to the local execution monitor at local database systems for executing.

For the purpose of coordinating the commitment of the Moflex transaction, the two-phase commit protocol is used by the mobile transaction manager. Commitment of the Moflex transaction occurs once its subtransactions that are managed by mobile transaction manager have reached one of the acceptable goal states, otherwise it is aborted. Once the local commitment of a compensable subtransaction takes place, the results are made visible to other transactions. In the case of non-compensable subtransactions, the last mobile transaction manager, which corresponds to the end location of the mobile host, plays the role as the committing coordinator.

In order to handle the mobility of transactions, the sub-transaction executed on the local database at the current mobile cell is split as the mobile host moves to another base station. Compensable and location independent subtransaction will be split into two transactions; one will continue and commit at the current local database, the second will be resumed at the new location. If the subtransaction is location dependent, at the new location, the subtransaction will be restarted. Non-compensable sub-transaction is restarted as a new one in the mobile cell if it is location dependent, otherwise it is continued.

4.2.8 Kangaroo Mobile Transaction Model

In [74], Kangaroo Mobile transaction model that captures both data and movement of mobile hosts has been proposed. Mobile transactions are initiated at
MHSs and entirely executed on the wired network. This model is based on open nested [75] and split transactions [62] and it satisfies most of the ACID properties. In this model, mobile transactions are called Kangaroo Transactions. Kangaroo transactions hop from one base station to another as the MH moves.

In this model, Data Access Agent (DAA) which is hosted at each base station is responsible for accessing data in the database. After receiving the transaction request from an MH, DAA forwards it to the base station that has the needed data. DAA is implemented on the top of existing global transaction manager. The base station, to which the MH is currently assigned, coordinates the mobile transaction execution.

The execution of a Kangaroo Transaction (KT) in each mobile cell is supported by a Joey transaction (JT) that operates in the scope of the base station. The Joey transaction plays role of a proxy transaction to support the execution of the subtransactions of the KT in the mobile cell. There is one Joey transaction for each base station. The mobility of the mobile host from one mobile cell to another is captured by the splitting the on-going Joey transaction at the old base station and creating a new Joey transaction at the new base station. The created Joey transactions are executed sequentially. Thus, all the subtransactions of JT1 are executed and committed before all subtransactions of JT2.

4.2.9 MDSTPM

In [76], MDSTPM (Multi-database Transaction Processing Manager Architecture) which proposes a framework to support transaction submissions from MHSs in a heterogeneous multi-database environment is presented. This execution model makes use of message and queuing facilities that manages the exchanges
between MHs and the wired multi-database systems. At each Fixed Host on top of existing local DBMS, a personal copy of MDSTPM exists. Local DBMSs takes care of local processing. The execution of global transactions is coordinated by MDSTPM. Because of MH disconnections, a FH coordinator is designated in advance. Hence, after MH has submitted a global transaction, it may disconnect and perform other tasks without having to wait for the mobile transaction to commit. On behalf of the MH, the coordinator host will manage the mobile transaction.

4.2.10 Preserialization

The preserialization model is presented in [77, 78]. In this model, MHs requests transactions from the multi-database systems where each DBMS runs on fixed hosts. Mobile transactions are considered as long-lived global transactions. Mobile transactions consist of compensatable subtransactions (called site transactions). The site transactions are allowed to commit independently of the global transactions so that effects of mobile transactions are minimized. This results in timely release of local resources. To verify the serializability of global transactions, a partial global serialization graph algorithm (PGSG) is used. The global transaction manager consists of global coordinator and a site manager layer. In the global layer, there exists a set of global transaction coordinators located at each BS and at any other node supporting external users. A set of site transaction managers at each participating DBMS are available in the local layer. MHs request global transactions using a global co-ordinator. Global co-ordinator in turn submits site transactions to local managers. The global layer is also responsible for handling disconnections and migration of mobile users.
4.2.11 Model for Supporting Mobile Collaborative Works

In [113], a transaction model for supporting mobile collaborative works is presented. This model supports sharing of information among mobile hosts which belong to a mobile affiliation work group. Once an MH is disconnected from other MHs in the mobile affiliation it will be removed from the mobile affiliation. This could happen due to low energy availability or the MH moves outside the communication range of the mobile affiliation.

Shared information among mobile hosts is fully distributed in a mobile sharing space. This model makes use of Export-Import repository which is a mobile sharing work space for sharing data states and data status. The physical export-import repository can be stored at one host or distributed among several hosts in a mobile work group. The data sharing among transactions in mobile environments is carried out by special types of transactions called export and import transactions that interact through an export-import repository. This way the data sharing can be carried out in both a synchronous and an asynchronous manner. A transaction that shares data or obtains data from other transactions is called delegatee or delegator transactions. The role of an export transaction is to support a delegator transaction to share its partial or committed results with delegate transactions and to transfer locks on shared data to delegatee transactions. Import transaction support a delegatee transaction at an MH to obtain needed information that can be either data states or data status from delegator transactions.

Data items are cached at MHs for disconnected transaction processing. To avoid data inconsistency among transactions at different MHs, the database server grants read or write locks on data items that are being downloaded in to the MHs. The
MHs which are disconnected from the servers will be able to join any mobile affiliation work group and share their data with other MHs. Two types of data sharing are distinguished: sharing data states and sharing data status. For sharing data values, the delegator transaction must hold a lock on the shared data item and only the value of the shared data item is revealed to other delegatee transactions. For sharing data status, a delegator transaction shares its locks on shared data to a delegatee transaction.

4.2.12 Transaction Models for MANET

In a mobile environment there are two architectures: General Mobile Computing Architecture and MANET architecture. In a General Mobile Computing Architecture, all MHs that are under the control of a Base Station roam within its cell. When an MH moves out of a cell and enters a new cell, it can no longer communicate with the previous cell’s Base Station and is under the control of the new cell’s BS. The communication between Base Stations takes place through a fixed network. The Mobile Ad-hoc Network (MANET) Architecture does not have a fixed infrastructure and fixed Base Stations. The MHs are connected through a wireless network with a frequently changing topology. All the reviewed models so far in this chapter are only based on the first architecture.

Transaction Management solution proposed in [118] uses MANET architecture adopted from group mobility model [93]. Two groups of MHs are classified in this architecture depending on communication capacity, computing power, disk storage, and memory size and energy limitation: SMH (Small Mobile Hosts) and LMH (Large Mobile Hosts). LMHs will store the whole DBMS and SMHs will store only some modules of the DBMS. The mobility of the submitting MHs as
well as the deadlines of the transactions are considered in the transaction management. It also aimed at reducing energy consumption at each MH by allowing each MH to operate in three modes, Active, Doze, and Sleep thus providing a balance of energy consumption among MHs.

Supporting database transaction services in MANET raises new issues. If an MH stores a database, then other MHs will try to submit transactions and get data from it. Since both the user and the data source will be moving in MANET, finding a route from one MH to another MH is necessary before a transaction is submitted. Moreover, many applications in this environment are required to be executed within their deadlines. Thus the Transaction Manager at the MH where the database is stored has to consider the mobility of the submitting MHs as well as the deadlines of the transactions. Another important issue in mobile ad-hoc networks is power or energy restriction on MHs.

In [119], the Mobile Nested Transactions (MNT) model for transaction processing coordination in groups of mobile devices is presented. This model supports distributed transaction processing over mobile devices. The main goal of MNT model is to move the work load currently performed at fixed network systems to mobile devices. In this way, MNT model achieves transaction processing independence in zones without having to access to a wired network or Internet. Using the MNT model, a transaction can be split into subtransactions that define logical work units thus creating a subtransactions hierarchical tree. Each subtransaction is processed by a host from the group of mobile devices. Once a subtransaction is successfully completed, changes get reflected in the database despite the failures of other subtransactions. Transaction processing (TP) monitors are responsible for coordinating distributed
transaction processing among different servers. This model is particularly useful for high-risk facilities such as oceanic platforms, ships at open sea, etc.

### 4.3 Concurrency Control Mechanisms in Mobile Database System

Database consistency is maintained through consistency preserving execution. The various schemes for serializing the execution of concurrent transactions in centralized and distributed environment were introduced in Chapter 2. In this section, we discuss the suitability of concurrency control schemes in the mobile environment.

The main reason for not considering the conventional concurrency control mechanisms in mobile environment is the performance problem that arises due to system overhead because of low capacity and limited resources. Modified conventional concurrency control mechanisms that are applicable in mobile environment are discussed below.

#### 4.3.1 Locking Based Concurrency Control Mechanisms

Of all the concurrency control schemes, most widely used scheme is two phase incremental locking with simultaneous release. There are three ways in which this scheme can be implemented in distributed computing environment: (a) centralized two phase locking (b) primary copy locking (c) distributed two phase locking.

**Centralized two phase locking:** In centralized two phase locking scheme, one node is responsible for managing all locking activities. The central node may always be available for locking activity in the mobile environment since the locking request traffic is very high. A mobile node cannot act as a central node because of its limited resources and mobility. Moreover, the maintenance of the status of (locked or free) of
data items is very difficult in a mobile unit. Since base station is a switch, it cannot take the roll of a central node. A fixed host cannot act as a central node since it is not equipped with a transceiver. Regardless of which component is chosen as a central node, the problem of single point failure cannot be avoided in this scheme.

**Primary Copy Two Phase Locking:** In this scheme, since the locking responsibility is distributed to many sites, the problem of single point failure is eliminated. Each lock manager is responsible for a subset of data items. The node executing the part of a transaction sends lock requests to appropriate lock manager. The choices for sites for the purpose of assigning locking responsibility in this scheme could be either base station or fixed host or both.

**Distributed Two Phase Locking:** This scheme is used to maximize the extent of lock distribution so that all nodes will serve as lock managers. But in the event of data partition, this algorithm could degenerate into a centralized two phase scheme. This scheme does not identify a suitable node which can act as a lock manager.

In conventional locking scheme, the communication overhead that arises due to locking and unlocking requests can create a serious performance problem. If a transaction executing in a mobile unit makes a lock request, it has to go through the following steps: (a) it sends a lock request to the lock manager site using a wireless message (b) the lock manager will decide to grant or to refuse the lock and send the result to the mobile unit using a wireless message (c) mobile unit makes a decision to continue with forward processing or block or roll back depending upon lock manager’s decision. Thus for each lock request, two wireless messages are required which would become expensive as the workload increases. Moreover, whenever there is a rollback, it generates additional overhead, once the transaction is restarted.
In order to maintain stronger degree of consistency, more resources are required compared to maintaining weaker degree of consistency. Thus by maintaining weaker degree of consistency, the cost is minimized to a larger extent. Weaker degree of consistency is acceptable in many data processing environments. This is more applicable in mobile database systems where mobile units are not likely to execute CPU intensive large update transactions. Instead, such transactions once initiated at mobile nodes, are executed at database servers with the strongest consistency level.

For getting more benefit, the concurrency control mechanisms that maintain a weaker level of consistency are not enough. This section discusses some of the concurrency control mechanisms which are specifically developed for mobile database systems.

**Distributed HP-2PL Concurrency Control Scheme**

A concurrency control scheme called Distributed HP-2PL [79] is based on two phase locking and it is an extension of HP-2PL [80] Concurrency control scheme. Conflict resolution scheme of cautious waiting [81] mechanism is used to minimize the degree of transaction rollbacks. For the purpose of managing the locking request for the data items available locally, each base station uses a lock scheduler in this scheme. A unique priority is assigned to each transaction which can be either the holder of the data item or the requester of the data item. Thus when there is a conflict between a requester and a holder, then the conflict is resolved using their associated priority and their executing status (committing, blocked, etc.).
4.3.2 Concurrency Control Scheme Based on Epsilon Serializability

In [82], a concurrency control scheme based on epsilon serializability (ESR) was discussed in which a limited amount of inconsistency is allowed. In order to generate an epsilon serializable schedule, this scheme makes use of two tier replication scheme [83]. The features of this scheme are: (a) it provides availability (b) it accommodates the disconnection problem (c) it is scalable (d) it reduces transaction commit time and number of transaction rejections. Using ESR approach, the amount of inconsistency is kept within a limit as specified by epsilon. When epsilon \( \rightarrow 0 \), ESR reduces to conventional serializability situation.

In order to achieve acceptable reduction in consistency, ESR [82, 84] is used in this scheme. An instance of ESR is defined by concrete specification of tolerated inconsistency. This concurrency control scheme can also be applied on fragmentable, reorderable objects [85], which include aggregate items such as sets, stacks and queues.

4.4 Commitment of Mobile Transactions

The mobility and other characteristic of MHs affect transaction processing especially its commitment. Some of the common limitations are: (a) An MH may cease to communicate with its BS for a variety of reasons. (b) it may run out to its limited battery power, (c) it may run out of its disk space, (d) it may be affected by security procedures, (e) physical abuse and accident, (f) it has limited wireless channels for communication and (g) unpredictable handoff.

Like conventional distributed database systems, a transaction in Mobile Database System may be processed by a number of nodes such as servers and MHs.
Therefore, some commit protocol is necessary for their termination. Conventional commit protocols such as 2PC, 3PC, etc., could be modified to work with mobile environment. However, they will not perform satisfactorily mainly because their resource requirements may not be satisfied by Mobile Database System on time. For example, the most commonly used centralized 2PC uses three message rounds in the case of no failure and uses five in the case of failure for termination [19]. It requires additional support (use of timeout) for termination in the presence of blocked or failed subtransactions. Thus, the time and message complexities are too high for Mobile Database System to handle and must be reduced to improve the utilization of scarce resources (wireless channel, battery power, etc.)

The mobility of MH adds another dimension to these complexities. It may force Mobile Database System to configure the initial commit setup during the life of a transaction [105, 105, 107]. For example, a proper coordination among the subtransactions of a transaction under participants-coordinator paradigm may be difficult to achieve with the available resources for its commitment. For example, a mobile unit may not receive coordinator’s vote request and commit message and it may not send its vote on time because of its random movement while processing a subtransaction. This may generate unnecessary transaction aborts. These limitations suggest that Mobile Database System commit protocol must support independent decision-making capability for coordinator and for participants to minimize cost of messages. A new commit protocol is required for Mobile Database System which should have the following desirable properties: (a) Minimum number of wireless messages should be used. (b) MH and Database Systems involved in transaction processing should have independent decision-making capability and the protocol should be non-blocking.
By analysing conventional commit protocols, it is found that timeout parameter could be used to develop a commit protocol for Mobile Database System. In conventional protocols, timeout parameter is used to enforce non-blocking property. A timeout identifies the maximum time a node can wait before taking any decision. The expiration of timeout is always related to the occurrence of some kind of failure. For example, in conventional 2PC the expiration of timeout indicates a node failure and it allows a participant to take a unilateral decision.

If a timeout parameter can identify a failure situation, then it can also be used to identify a success situation. Under this approach the end of timeout will indicate a success. The basic idea then is to define a timeout for the completion of an action and assume that at the end of this timeout the action will be completed successfully. For example, a participant defines a timeout within which it completes the execution of its subtransaction and sends its update through the coordinator to Database Systems for installing it in the database. If the update does not arrive within timeout, then it would indicate a failure scenario. The coordinator does not have to query the participant to learn about its status.

Recently timeout parameter has been used in a nonconventional way for developing solutions to some of the mobile database problems. A commit protocol known as Transaction Commit Timeout (TCOT) which uses timeout parameter to indicate a success rather than a failure was proposed in [105].

**4.5 Caching in Mobile Environment**

In a Mobile computing environment, instead of storing data in a central location (server), data is being moved closer to applications. This enables data processing to be performed more efficiently and autonomously. Because of the
distributed nature of mobile environment there are several constraints that make finding a solution to this problem a difficult one: (a) Limited resources (b) Limited energy (c) Disconnection and (d) Bandwidth asymmetry. Limited CPU power and memory capacity will not allow an MH to perform even simple operations on local data. It is infeasible to cache the entire database in an MH due to limited storage capacity. In a Mobile Database System, Mobile hosts move across wireless cells and access a centralised database server. Minimising the amount of data transmitted between the server and the MH over wireless link is a common objective in mobile database system [86].

Today, it is common for mobile database applications to query a central database directly. Direct (synchronous) calls to a database guarantee a consistent up-to-date view of the database. However, they require full connectivity, constrain mobility, and consume a lot of battery power. The cost involved in accessing central databases for every operation is quite high. Moreover, frequent central database access can impact response time and throughput of a mobile application. To solve this problem, the common approach is to cache data locally in an MH and thereby increase the performance of local queries. But, it may require a more complex cache maintenance schemes. Maintaining consistency of cached data is a real challenge on a resource constrained, nomadic, frequently disconnected mobile device, while minimizing energy usages.

4.5.1 Mobile Database Caching

In conventional caching schemes, both strong connectivity and symmetric bandwidth are assumed [87]. The characteristics of mobile environment such as disconnection and asymmetric bandwidth, necessitates caching for maintaining
mobile computing service. In order to reduce wireless bandwidth requirement and to cope with disconnection, caching of frequently accessed data is considered as a useful strategy [86, 88, 89, 90, 91]. However, the use of mobile caching systems raises the challenge of ensuring that transaction semantics are not violated as a result of the creation and the destruction of cached data. In [92] these algorithms are defined as transactional cache consistency maintenance algorithms. These algorithms can be divided into two categories: (a) Avoidance-based where servers either send invalidation messages to signal that a cached item must be updated or, depending on usage patterns, send the update itself. (b) Detection-based where clients send queries to the database server to validate cached data. Avoidance-based approaches have largely dominated mobile database literature.

Consistency is enforced in avoidance based approaches in such a way that it is impossible to ever access stale data. Database servers ensure consistency by directly manipulating the contents of clients’ cache. This is achieved by means of broadcasting of updates [94, 95], invalidation reports and invalidation messages [96, 97, 86].

4.5.2 Invalidation Report Caching Schemes

The scheme in which database updates are broadcast naively is considered as an impractical scheme. Invalidation messages can be sent to MHs caching particular data. In this scheme, it is essential that the state of each MH is maintained by the server. If MHs are disconnected, at reconnection time, they must connect to the server to receive any pending invalidation messages. In this scheme, the MH cache consistency is maintained and communication between client and server is minimized.
But scalability problem arises due to the fact that the state of each client’s database is to be maintained by the server.

Scalability problem is solved by broadcasting invalidation reports. Invalidation reports take care of the possibility of disconnection by MHs and contain aggregate information about data items that have changed. The need for the server to maintain the state of each MH is removed by the use of invalidation reports. Invalidation reports are broadcast to all MHs and reports itself represents the state of data that could be cached at each MH. Upon receipt of invalidation report, MHs compare the set of invalidated objects to the objects in the cache. Invalidated objects are evicted or refreshed if needed. Generally each invalidation report carries a timestamp from its issuing service. The validity of an MH’s cache is taken from the last invalidation report. Cache invalidation using invalidation reports has been implemented in [90, 98, 99].

There can be two broadcast schemes which are based on the timing of the reports: (a) Synchronous scheme [86, 100, 101] in which broadcasting of cache invalidation reports is carried out periodically. Synchronous servers aggregate updates over a fixed period of time and then broadcast these invalidation reports with the time of the updates all in one message. The drawback to this approach is that latency is introduced between the time of the update and the invalidation broadcast. (b) Asynchronous scheme in which [102, 103, 100, 99] the database server broadcasts an invalidation report as soon as an item in the database changes. The biggest problem with this approach is the unpredictable amount of time a client must wait for an invalidation message.
There are many tradeoffs when designing systems that use invalidation reports. These tradeoffs are discussed in detail in [86] that served to motivate subsequent research. One obvious research direction was to minimize the size of the invalidation report. In [104], an invalidation scheme called Invalidation by Absolute Validity Interval (IAVI) is presented. This method exploits pre-defined patterns in how data changes. The predicted period of validity of a data item is defined to be the Absolute Validity Interval (AVI). The AVI is distributed to mobile clients as invalidation. A mobile client can verify the validity of a cached data item by comparing the last update time with the AVI. The client invalidates items in its cache if the current time exceeds the validity period of the AVI. The use of AVIs allows mobile clients to estimate the validity of its cache during periods of disconnection. For example, suppose a database is known to update hourly with new price information. AVIs on data allow a mobile sales agent that is disconnected between updates to query its database cache with a reasonable certainty that the information is valid. When the mobile client is connected, the database server sends invalidation reports to inform clients of changes to AVIs rather than data item itself. Performance studies show that the use of IAVI significantly reduces the mean response time and invalidation report size over previous bit-sequence [90] and timestamp [86] mobile cache invalidation schemes.

Concurrency control without locking in mobile environment is presented in [114]. This approach makes use of AVI (Absolute Validity Interval) to enforce concurrency control. But it calculates AVI only based on previous update interval. In [52], a method based on PLP (Predicted Life Period), which takes care of the dynamicity of the life time of data, has been proposed. In this architecture, the base station is used for storing a cache. The cache manager module monitors and
coordinates the operations in the cache. Database server contains data that can be accessed by mobile clients and fixed hosts. Here, life span of data is predicted based on the probability of updation of data item. In this method, PLP of data item is very close to the actual valid life span of a data item. But this approach does not support disconnected operations. Concurrency control mechanism used in this research work is based on PLP.

In [115], a technique for composing near optimal invalidation reports using bit-sequence algorithms [90] is presented. This work follows earlier work on optimal invalidation report construction [116] and is based on the bit sequence approach presented in [117]. Two techniques are commonly used to reduce the size of an invalidation report: bit-sequence mapping and update aggregation. Update aggregation is often used in conjunction with bit-sequence algorithms to reduce the number of timestamps in a report. Instead of having one timestamp for each updated item, the report uses only one timestamp, which is the time when the report is to be delivered. Update aggregation is successful in reducing the size of the report. However, the accuracy of the report is reduced and valid items count is falsely invalidated. In [115], the problem of false invalidation on cached items when there is insufficient time detail in the invalidation report is considered. In this approach they exploit clients’ disconnection patterns (i.e. the distribution of the time spans between disconnection and reconnection) to reduce the rate of false invalidations. They analyze the relationship between false invalidations and reconnection patterns and formulate a technique that reorganizes the hierarchy of bit-sequences to take into account clients’ reconnection patterns. Through simulation they show that their technique has significant improvements over [116], but in practice the technique has an enormous overhead and does not scale to many clients.
4.6 Summary

From the review of various transaction models, one can understand how transaction processing has evolved from centralized and distributed systems to transaction processing over mobile devices. Traditional transaction models were designed to support transaction management in centralized and distributed environment. In order to cater to the requirements of mobility and disconnection, many mobile transaction models were also developed.

Since Kangaroo model assumes the autonomy of the underlying DBMS systems, subtransactions are allowed to commit/abort independently. By means of compensating transactions, Atomicity may be achieved. While the Semantic approach allows processing anywhere in the mobile platform, it is a restricted type of processing in that only one server is assumed and all fragments processed at the MU must be returned to the server prior to commit. All but the Semantics-based approach may violate durability. This is because local transactions which have committed may later be “undone” by a compensating transaction. All models but Reporting assume that mobile transactions are requested from MHs. In Reporting, transactions can be requested by any host. The kangaroo is assumed to execute only on the fixed network. This really does not apply to the Reporting approach and Co-transaction approaches. The Semantics model assumes that the execution at a server on the fixed network is limited to the creation and then update of the fragments. Clustering and MDSTPM are assumed to execute either at MH or Fixed Network.

When simultaneous access to data is made at the server, concurrency control techniques are employed to avoid data inconsistency. Conventional locking based
concurrency control methods like centralized Two Phase locking and distributed Two Phase locking are not suitable for mobile environment. In centralized two phase locking scheme, where one node is responsible for managing all locking activities, the problem of single point failure cannot be avoided. The distributed two phase locking scheme allows all nodes to serve as lock managers. But in the event of data partition, this algorithm could degenerate into a centralized two phase scheme. In conventional locking scheme, the communication overhead that arises due to locking and unlocking requests can create a serious performance problem because of low capacity and limited resources in mobile environment. Moreover, it makes mobile hosts to communicate with the server continuously to obtain and manage locks.

The transaction model proposed for supporting mobile collaborative works uses Export-Import repository which is a mobile sharing work space for sharing data states and data status. In this model, locking is used as the main technique for achieving concurrency control. As a result, disconnection of a mobile host or a transaction failure will result in blocking of other transactions for a long period. The transaction management solution which has been proposed for MANET architecture is for reducing energy consumption at each MHs. It allows each MH to operate in three modes i.e. Active, Doze and Sleep thus providing a balance of energy consumption among MHs. For enforcing concurrency control, a method based on PLP (Predicted Life Period) has been proposed to take care of the dynamicity of the life time of data.

In the case of commitment of Mobile transactions, local commitment at the MHs is followed by final commitment at the servers. When an MH is disconnected, mobile transactions can commit locally and the results are made available to other local transactions at the same MH. Upon reconnection, these results are reconciled.
with the data at the server. Abortion of locally committed transactions will result, if there is any inconsistency.

4.7 Discussion

After careful examination of the literature, the following issues are identified in mobile database transactions.

1) **Disconnection issue:** Most of the existing frameworks do not attempt much on disconnection issue.

2) **Priority based transaction scheme:** In a mobile computing environment, there is always a need to schedule transactions based on energy availability and connectivity in order to minimize transaction abortion rate. This issue is not taken up by the most of the researchers so far.

3) **Concurrency control without locking:** Most of the traditional techniques available in mobile environment achieve concurrency control by using the concept of locking. This may not be feasible in mobile environment due to variable bandwidth and frequent disconnections etc. In order to enforce concurrency control, lockless mechanisms which minimize communication overhead and enhances transaction throughput are more attractive. But few literatures are discussing these aspects that too at a preliminary level. A complete Agent based architecture with lockless mechanisms for concurrency control is not found in the literature.

4) **Fixed / Mobile Agents:** The use of Fixed / Mobile agents in which frequently accessed data are cached is absent in many related works. A complete Agent based architecture is not proposed so far.
5) **Energy Conservation:** It is a challenge in mobile environment. By conserving energy, there is always a scope for allowing long-lived mobile transactions. The reviewed techniques do not address the concept of Energy Conservation (EC) mode.

### 4.8 Problem Statement

Performing transaction processing in mobile environment is identified as a critical research problem. In this thesis the above research problem is taken for investigation. Based on the issues outlined in the previous section, an agent based transaction management framework is proposed in which mobile users can simultaneously access the cached data from the Fixed / Mobile Agents even when the Mobile Hosts are disconnected from the network.

The resultant framework shall meet the following requirements:

- Agents availability
- Multi-hop connectivity support.
- Transaction completion at finite time interval.
- Minimum response time with low transaction abortion rate.
- Priority based on energy availability.
- Priority based on connectivity strength.

### 4.9 Proposed Methodology

The issues concerned with mobile database transactions and mobile applications are identified through a detailed literature review. Traditional transaction models are investigated to find out their suitability for applying them to mobile
computing environment. Existing mobile transaction models are also investigated and their strengths and weaknesses are observed and suitable changes are incorporated.

The following methodology is adopted to develop an enhanced transaction management framework.

(i) A generic mobile transaction model has been proposed by making suitable modification in the existing models.

(ii) New transaction processing algorithms are proposed by considering each of the above requirements such as optimal response time, low transaction abortion rate, priority based on energy availability and connectivity etc.

(iii) A common simulation test bed is proposed to validate the above framework.

(iv) The proposed framework has been analysed for performance and results are compared with existing frameworks for performance improvement.

(v) Based on the results and analyses, best strategy transaction scheme is identified which can be proposed as a general model for Mobile Transaction Processing.

The components and requirements assumed for this research are listed below. The proposed schemes have few common components at the architecture level. The performance analyses of all the four schemes are to be evaluated under the same system environment and mobility model.

- **Components:** The common components used in the Mobile transaction architecture models are Mobile Support Stations (MSSs), Mobile Hosts (MHs), Data Base Servers (DBSs), Fixed Agents (FAs) / Mobile Agents (MAs), and Base Station Controller (BSCs).
- **Requirements:** The proposed schemes are implemented by using the following hardware, software and mobility model.

  (i) **Hardware:** Pentium Dual Core processor based PC @ 2.4 GHz with 3 GB RAM and 320 GB HDD.

  (ii) **Software:** Java 2 and Network Simulator – 2 (NS2 Version 2.34)

  (iii) **Mobility Model:** Random walk mobility model [20] has been identified as a basic mobility model from the few literatures related to this work. However, other mobility models can also be used. As the main objective of this research work is to evaluate the performance of transactions under mobile database architecture, the criteria of performance evaluation under different mobility models has not been considered in this research.

  Based on the above methodology and requirements, four schemes are proposed in this thesis. These four schemes are discussed in the following chapters.