A Mobile Transaction System for Open Networks

R. B. Patel[1] and **Anu**[2]

Abstract - ***The evolution of mobile computing devices and wireless network has created a new mobile computing environment. Users equipped with portable devices can access, retrieve and process information while in mobility. Mobile devices like laptops; mobile phones have become more powerful data processing elements. Traditional transaction model has moved forwarding to mobile transaction system. Autonomous decentralized systems represent examples of environments for which the use of mobile codes is quite convenient. For example, designing highly scalable distributed systems in a massive, heterogeneous and multi organizational distributed environment seems to benefit much from mobile codes, given their ability to decentralize processing; to adapt to the autonomy of systems; to flexibly allow the management of installed code; and their support to the interaction with human users. This paper presents a hierarchical transaction model for the execution of distributed transactions with mobile code on open networks. The developed transaction model is built upon the concept for fault tolerance of mobile code based executions. The presented transaction model is an open nested transaction model. This model supports those parts of a distributed transaction which is executed asynchronously in relation to other parts of the same global transaction. Furthermore, the model is able to recover the execution of a transaction when a sub-transaction of this transaction becomes unavailable for a long period of time and the results of a comparison of developed model, with some existing ones, are also reported. We have also suggested and implemented an efficient naming and locating mechanism for tracing/finding the status of a transaction whenever fault(s) arises in the transaction processing system/network or processing of a sub-transaction is delayed.***

Index Terms - MH, Transaction, ACID, TPS.

1. INTRODUCTION

Technological advancements in networking and distributed processing are enabling the emergence of new types of distributed processing environments. Exemplified by electronic service markets or virtual enterprises, such environments are highly complexed distributed systems that support corporations needs for integrating systems and that allow new forms of automated cooperation. Many types of mobile computing devices such as laptops, personal digital assistants (PDA) are available. The capacities of these mobile devices have become more powerful. They have more processing speed and longer operating time. Mobile computing devices are becoming the major work processing equipments in daily activity. Combining with the expanding of the high-speed network like the Internet, mobile computing applications are growing rapidly. Some of the characteristics of such systems are: they are composed by a multitude of autonomous organizations cooperating or competing to achieve their own goals; massive geographical distribution; they encompass a huge diversity of types (qualities) of communication links; the execution of inter-organizational activities are typical in such environments; a multitude of services are offered to a multitude of clients of such services; different type of services exist and they may range from totally automated services to services executed by human beings, high dynamism with no global control, high heterogeneity and coexistence of different types of hosts such as laptops, personal computers, powerful workstations or mainframes). Environments with these properties will be called here open networks, i.e., Internet.

[1,2]Dept of Computer Engineering, M. M. Engineering College, Mullana-133203, Haryana, INDIA
E-Mail: [1]patel_r_b@indiatimes.com and [2]anurawal2k@yahoo.com

A transaction is a collection of operations on the physical and abstract Application State [1, 28, 29]. Transaction processing systems provides a means to record all states and effects of execution of program in computing resources. Transaction not only relates to operation on database system but also involves in many daily applications upon many computing resources like telephone call, email system, flight reservation. Mobile transaction is more complicated than conventional transaction in both of design and execution states. When a mobile host (MH) moves from one region to another, many computing activities like establish new communication channel, forward the state of transaction to new base host (BH) [15] are involving. The execution of mobile transactions is not only unpredictable in time but also depends on their location.

A computation processing is considered as a transaction or conventional transaction if it satisfies ACID [1] [2] (Atomicity, Consistency, Isolation, and Durability) properties.

1. Atomicity: an executable program, assumed that this program will finally terminate, has one initial state and one final state. If it appears to the outside world that this program is only at one of these two states then this program satisfy the atomicity property. If there are intermediate results or message needs to be displayed, then they are not displayed or they are displayed in final state of the program. If the program achieves its final state it is said to be committed, otherwise if it is at the initial state after some execution steps then it is aborted or rollback.
2. Consistency: if a program produces consistent result only then it satisfies the consistency property and it will be at the final state or committed. If the result is not consistent then a transaction program should be at the initial state, in other word the transaction is aborted.
3. Isolation: if a program is executing and if it is only process on the system then it satisfies the isolation property. If

there are several other processes on the system, then none of the intermediate state of this program is viewable until it reaches its final state.

4. Durability: if a program reaches to its final state and the result is made available to the outside world then this result is made permanent. Even a system failure cannot change this result. In other words, when a transaction commits its state is durable.

A mobile transaction is a set of relatively independent (component) transactions which can interleave in any way with other mobile transactions. A component transaction can be further decomposed into other component transactions, and thus mobile transactions can support an arbitrary level of nesting.

Let us assume that S is a two level mobile transaction that has N component transactions, $T_1, \ldots, T_N$. Some of the components are compensatable; each such T_J has a compensating transaction cmst_T_J that semantically undoes the effects of T_J but doesn't necessarily restore the database to the state that existed when T_J started executing. Component transactions can commit without waiting for any other component or *S* to commit, i.e., component transactions may decide to commit or abort unilaterally. However, if *S* aborts, a component transaction that has not yet committed is aborted.

Mobile transactions, components or otherwise, are distinguished into 4 types:

a) Atomic transaction: these are associated with the significant events {Begin, Commit, Abort} having the standard abort and commit properties. Compensatable and compensating transactions are atomic transactions with structure- induced inter-transaction dependencies. A compensatable component of S is a component of which can commit its operations even before S commits, but if S subsequently aborts, the compensating transaction cmst_T_J of the committed component T_J must commit. Compensating transactions need to observe a state consistent with the effects of their corresponding components and hence, compensating transaction must execute (and commit) in the reverse order of the commitment of their corresponding components.

b) Non- compensatable transactions: these are component transactions that are not associated with a compensating transaction. Non- compensatable transactions can commit at any time, but since they cannot be compensated, they are not allowed to commit their effects on objects when they commit. Non- compensatable transactions are structured as sub transactions (as in nested transaction) which at commit time delegate all the operations that they have invoked to *S*.

c) Reporting transactions: a reporting component T_J can share its partial results with *S*, i.e., a reporting component delegating some of its results at nay point during its execution. Whether or not a reporting component delegates all the operations not previously reported to *S* when it commits depends on whether or not it is associated with a compensating transaction.

d) Co-transactions: these components are reporting transactions that behave like co-routines in which control is passed from one transaction to another at the time of sharing of the partial results, i.e., co- transactions are suspended at the time of delegation and they resume execution where they were previously suspended. Thus, as opposed to non-compensatable transactions, co-transactions retain their state across executions; and as opposed to reporting transactions, co- transactions cannot execute concurrently.

Compensatable and non- compensatable components can be further considered as a vital transaction in that S is allowed to commit only if its vital components commit. If a vital transaction aborts, S will be aborted. Transaction *S* can commit even if one of its non- vital components aborts but S has to wait for the non- vital components to commit or abort.

The simplest form of transaction is flat transaction. A flat transaction can be considered as a sequential correctness computer program. Every execution step is after one another. Flat transaction has many disadvantages. For example, it cannot support long transaction efficiently. If failure happens during its execution then it has to rollback to its initial state and wastes all useful computation. Nested transaction model is a more flexible transaction model. This model is a tree of transactions. A big transaction is refined into many smaller (flat) transactions called sub-transactions. These sub-transactions can execute concurrently in different processes in different processing hosts. The ACID properties are more relaxed in this nested transaction model. Autonomous decentralized systems represent examples of environments for which the use of mobile codes [20] is quite convenient. For example, designing highly scalable distributed systems in a massive, heterogeneous and multi organizational distributed environment seems to benefit much from mobile codes, given their ability to decentralize processing; to adapt to the autonomy of systems; to flexibly allow the management of installed code and their support to the interaction with human users.

In this paper we present a model for the execution of distributed transactions with mobile code on open networks. The developed transaction model is built upon the concept for fault tolerance of mobile code based executions [20]. The presented transaction model is an open nested transaction model. The model supports those parts of a distributed transaction which are executed asynchronously in relation to other parts of the same global transaction. Furthermore, the model is able to recover the execution of a transaction when a sub-transaction of this transaction becomes unavailable for a long period of time. Open nested transaction model has been proposed for coping with long running activities and with the autonomy of systems in multi databases and thus take into consideration aspects of open networks. We have also suggested and implemented an efficient naming and locating mechanism for tracing/finding the status of a transaction whenever fault(s) arises in the transaction processing system/network or processing of a sub-transaction is delayed.

The rest of the paper is organized as follows. Section 2 gives overview of the currently agreed mobile computing environment. Section 3 discusses some of the limitations of exiting transaction models. Section 4 describes issues in Mobile Transaction Processing. Section 5 presents System Model. Section 6 gives model of the Transaction Processing System (TPS). Transaction model is presented in Section 7 and Section 8 gives implementation and performance study compared with the existing one. Related works is presented in Section 9 and conclusion of this work is given in Section 10.

2. MOBILE COMPUTING ENVIRONMENT

It is important to identify and define the mobile computing environment. Based on that defined mobile environment, requirements as well as characteristics will be identified. Mobile computing environment includes: a wired network with fixed work-stations or fixed hosts (FH), mobile hosts (MH) and base host (BH) [15] which is similar to mobile support stations (MSS) [3] [4] [8] [9] as shown in Figure 1.

Connection between MH and BH is wireless network; this network has characteristics low bandwidth, error-prone and frequently disconnection. These characteristics are discussed in detail in the next subsection 2.1. BH and FH communicate with each other via reliable high speed connection networks, which can be wired or wireless network within limited range, such as inside a building. The BH is motionless. MHs can include broad types of mobile devices, typically laptop computers with high-speed modems. Works can be shared between MH and FH. The role of BH is not only as processing element but also it is acting as an interface to help MH getting contact with relevant FH.

Each BH is being responsible for all the tasks which occur in a region. One MH can only connect to one BH at any given time but at overlap region during the handoff it connects to two BHs. A MH is moving from one region to another when computation task is in processing, and sometimes MH requests to connect to a database or computing resource resided from a FH on fixed network. This work will be done with the help of BH. The BH will receive requests from MH, forward the requests to the responsible FH and return the answer from the FH to the MH. When a MH is leaving a region controlled by a BH, this BH will perform a handoff operation to transmit or forward all information related to this MH to next BH. The next BH in new region will be ready to support the MH.

Databases and other computing resources are stored on the FH or wired network, this environment allow mobile environment inherits from the current existing distributed computing environment. Normally, power supply and storage device limits MH computing capacity.

However, with the current technology, the power of mobile computers can last for several hours and the storage devices can store a large amount of data [5]. Then MHs can become major hosts for data processing.

The main features of mobile computing environment are communication, mobility, portability [6] and heterogeneity [3]. There are many research issues that are arising from these features.

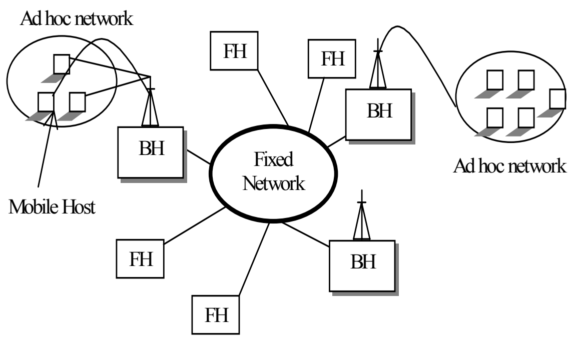

Figure 1: General Architecture of Mobile Computing Environment

2.1 Communication- MHs are connected with BH through wireless network. It is obvious that this wireless network does not have capacity as fixed wired network. First, the wireless bandwidth is very low, for example cellular network has bandwidth in the order of 10Kbps or wireless local area network has bandwidth of 10Mbps- 1000Mbps [5], Second, wireless network are having high error rate and frequent disconnection [3] [5], the same network data package may require retransmit many times. When MH is moving from one region to another, the current connection with BH will need to be changed to new connection. This process requires two steps: disconnecting from the current connection and establishing a new connection. The above disadvantages result in taking more time to transfer a same amount of data from the MH to FH and vice versa. Retransmit data causes unnecessary processing power, which is already very limited on the MH. The situation is more complicated if two MHs need to exchange data during cooperative task. Messages cannot be delivered directly between two MH but can be via one or more BH. Because of larger overhead in communication time, the longer time requires for MH to perform computation. Caching mechanism is currently the major method to ease the problem.

2.2 Mobility- Mobility is the most frequent activity of a MH. When MH is moving from one region to another in wireless network, the connection will need to be changed because one BH can only support MHs within its limited area. This cause frequently need of reconfiguration network topology and protocols. The more mobility causes the more time spends on reestablish communication between MH and BH. Because the activities of MH need support from its BH, therefore location management is another problem caused by the mobility of MH. MHs need to track BH in order to obtain data from the FH or other MH. In other hand, BH also needs to keep track on MH in order to transmit the result from the FH or to update the state of current MH profile. Mobility of MH raises the question on location dependent data [10]. The same query will have different results depending on the location of MH.

2.3 Portability- The availability of mobile devices depends on their power supply. A mobile phone can live up to five days but the laptop can only be for few hours. The more complicated application requires more processing power. Refining computation process into smaller partition (fined grain) or shifting heavy process from MH to FH for processing can save energy. Communication in MHs requires a lot of power. Compressing data or data distilling before transmission can reduce communication time. Caching also help MH tasks in disconnected period. Portability of MHs requires more sophisticated software applications. MH has smaller user interface like display screen, keyboard [6]. Many PDA support handwriting, therefore handwriting recognition software is required.

2.4 Heterogeneity- One BH needs to support broad types of mobile devices which operate in its region. Identifying what kind of hardware of the MH is important. Different MH requires different applications and data representations. When MH requests communication with other MH, the heterogeneous problem needs to be taken into account. How does BH solve this problem? A Composite Capabilities/Preference Profiles (CC/PP) can be used to provide a description of mobile device [14]. Different BH are in different heterogeneous network and these BH need to cooperate and communicate with each other for exchanging data. A standard interface is needed between BH. Java technologies or a middleware like CORBA [16] can be used to solve the heterogeneous problems.

3. LIMITATIONS OF EXISTING TRANSACTION MODELS

A computation that accesses shared data in a database is commonly structured as an automatic transaction in order to preserve data consistency in the presence of concurrency and failures. However, a mobile computation that accesses shared data cannot be structured using atomic transaction. This is because atomic transactions are assumed to execute in isolation that prevents them from splitting their computation and sharing their states and partial results. As mentioned above, practical considerations unique to mobile computing require computations on a MH to be supported by a MSS for both communication and computation purposes. This means that a mobile computation needs to be structured as a set of transactions some of which execute on the MSS.

In addition, Mobile computations are expected to be lengthy due first to the mobility of both data sources and data consumers, and second to their interactive nature, i.e., pause for input from the user. Thus, another requirement of mobile computations that atomic transactions cannot satisfy is the ability to handle partial failures and provide different recovery strategies, thus minimizing the effects of failures.

Nested transactions [21], where a (parent) transaction spawns (child) transactions, provide some more flexibility than atomic transactions in supporting both splitting of their computation and partial failures. However, nested transactions do not share their partial results while they execute. Nested transactions support procedure-call semantics and commit in a bottom-up manner through the root, i.e., when a child transaction commits, the objects modified by it are made accessible to its parent transaction while the effects on the objects are made permanent in a database only when the root transaction commits. This also means that the state of the mobile computations must be retained until the root transaction completes its execution. Consider the case in which the root executes on the MH whereas the child transactions execute on the MSS. If sub transactions do not retain their state after completing their execution, then the state of the whole computation needs to be maintained at all times on the MH in spite of its limited resources. On the other hand if sub transactions retain their state, the state of the computation is spread among MSS along the path of the MH making atomic commitment expensive.

Open- nested transactions such as Sagas [24], Split transactions[22] and Multi-transactions[23] relax some of the restrictions of nested transactions by supporting adaptive recovery, i.e. , allowing their partial results be visible outside a transaction. This is because, in open nested model, component transactions may decide to commit or abort unilaterally. It is interesting to note that most open- nested transaction models have been proposed in the context of multi database systems. A mobile database environment can be viewed as a special multi database system with special requirements. For example, the notion of local autonomy in mobile environments is manifested in the ability of the MHs to continue to operate in an independent fashion when they are disconnected.

Yet two specific requirements of transactions in mobile environment cannot be satisfied by current open transaction models. First, the ability of transactions to share their partial results with each other while in execution, and second to maintain part of the state of a mobile computation on a MSS in a way that minimizes the communication delays between a MH and MSS.

4. ISSUES IN MOBILE TRANSACTION PROCESSING

Mobile transactions are long-lived, bound to many different types of mobile devices, involved in heterogeneous database and network and execution time is varying. This section focuses research challenges in mobile transaction mainly on mobile database, service handoff and scheduling.

4.1 Mobile Database

Currently, the mobile transaction is developed on the top of currently existing database system. Most of mobile transaction models are based on the earlier discussed mobile environment. In this environment, the database resides, replicated and distributed on the fixed hosts in wired network. However, the capacity of mobile computing device is expanding and a MH can become a host for data processing or a place to store the native data. In this case, the physical location of database system is changing. Identify the location of the MHs which stores the required data is one of the major issues in mobile database [5]. To obtain optimization on query processing, databases are replicated or fragmented in MH. Because of the disconnection and mobility of MH, maintaining data consistency between MH is more complicated. Location dependent data also needs to be considered.

4.2 Service Handoff

When a MH moves into a new region, a new BH is assigned to this MH. Information about current transaction state is saved and transferred from old BH to next BH. This operation sometimes is unnecessary because not all the time MH requires assistant. Figure 2 illustrates the situation. MH *M* is moving from region A to region C through region B. However, in region C the MH does not need any assistant from BH in region B. The information about transaction state should directly forward to BH in region C. This information package also includes the hardware profile of MH, context of application and environment. If this information is stored at MH then the MH can become an active element, which can initiate a connection when needed. The question is how a MH finds out what BH it should connect to. Currently, when a MH wants to exchange information with another MH then it has to rely on the support from at least one BH. How can one MH directly obtain communication channel with other MH?

4.3 Scheduling

Execution time of mobile transaction is varying. Mobile transaction can easily miss its required deadline due to its mobility and portability. It is not applicable in mobile transaction if a missing deadline transaction is always aborted. Missing deadline causes inconsistency in global state of transaction and blocks other transaction's execution. Enforcing technique like earliest-deadline-first [3] can be applied. Mobile transaction requires flexible scheduling mechanism. Scheduling a transaction in a FH is different from MH. Schedule in mobile transaction should take into account the mobility of MH in both location and time. MH should be able to reschedule its execution plan according to its physical state (power, communication bandwidth).

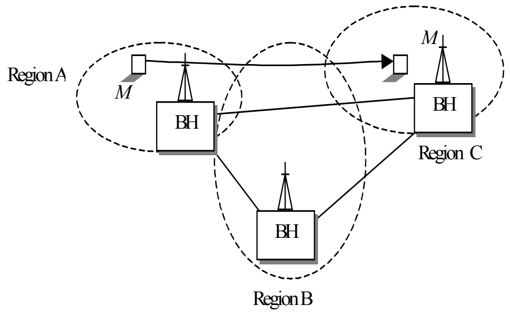

Figure 2. Service Handoff between BH

4.4 Caching

Caching of data at MUs can improve performance and facilitate disconnected operation. Much research has been performed in the area of MU caching [26]. Caching issues are complicated by the use of Location dependent data (LDD). Because of the fact that data which is cached can be viewed as a temporal replica of spatial data, as a MU moves into new data regions the cached data may become obsolete. This data is not stale because it is incorrect, but may not be desired because it is from a foreign region. Replacement policies need to be re-examined to include location information. For example, data from a foreign region should perhaps be replaced before data from the current home region even though the foreign data is more recently used. However, this is further complicated by the fact that ongoing or future queries could be bound to foreign regions. The MU mobility is such that the MU could very quickly move back into the home region for this data, making the re- placement policy also subject to movement of the MU. All of these issues are beyond the scope of this paper, but certainly need to be studied.

5. SYSTEM MODEL

A transaction submitted from a MH is called mobile transaction [3]. The MH, which issues transaction, and the MH, which received the result, can be different. For example, a user queries for a bus timetable from its laptop and requests the answer will send to mobile phone via SMS. A MH is a mobile computer which is capable of connecting to the fixed network via a wireless link. A FH is a computer in the fixed network which is not capable of connecting to a MH. A BH is capable of connecting with a MH and is equipped with a wireless interface. BHs, therefore, act as an interface between MH and FH. The wireless interface in the BHs typically uses wireless cellular networks because of the characteristics of mobile environment; mobile transaction has several additional requirements:

1. As MH has less processing capacity as FH, so mobile transaction should be able to split into a set of smaller transactions. These shorter sub-transactions can execute on FH or other MH. If possible, most of the computation on the MH should be shifted to FH for processing. When computing tasks are moving to FH, the FH have more computing power and shorter processing time. In addition, the computing resources are closer in FH. If the tasks require extra computing resources, wired network bandwidth is faster for resource allocating than wireless network. MH can save energy by disconnecting their connection while waiting for the results from the FH.
2. Mobile transaction has longer processing time or long-lived. Because of the communication overhead and frequent disconnection, the time required for exchanging needed data between MH and BH is longer. A part from this, MH has slower processing speed therefore a same transaction on MH will require longer time for completing than on the FH.
3. Mobile transaction should be executable when MH is in mobility and disconnected from the computing resources. It is not possible for MH staying connected all the time with the data resources. After the needed data has been caching into mobile storage device then MH can operate in autonomous mode. Data inconsistency in short time should be allowed. When the connection is established the new data item will be updated to the main database.
4. Mobile transactions require being able to operate in distributed heterogeneous environment. Different types of MH cooperate in mobile environment and different database systems are accessed during execution state of

mobile transaction. Mobile application should take into account the representation of data format in different system.

Mobile transaction is a collection of BH which traps a transaction in a region. For this purpose it runs a region manager (a transaction processing system in a region is called region manger), other nodes in this region may be mobile or fixed, i.e., network may be ad-hoc/ fixed. The BH works like a manager when a transaction leaves from a network domain, it may be managed by domain manager server (DMS). A DMS is basically a transaction processing system.

Some of the techniques developed in conventional transaction such as two phases commit (2PC) protocol, caching mechanism is needed to be extended or modified to be able to apply in mobile transaction. Another issue is to make the intermediate states of mobile transactions available to others. This will release locks on data item earlier and avoid blocking other transactions. DMS looks apart BH and BH looks apart a set of nodes lying in an inter-network, the system is a hierarchical.

6. TRANSACTION PROCESSING SYSTEM FOR DISTRIBUTED DATABASES

A *transaction processing system* (TPS) uses the transaction as a basic unit of task. It typically consists of a transaction manager, resource managers, and clients as shown in Figure 3. Client applications start particular transactions, within whose scope they forward data requests to registered resource managers, and commit or abort the transactions. *Resource managers* are entities, which store and manage data objects manipulated by transactions. They ensure durability of transactions. A database system is an example of a resource manager. The *transaction manager* enables clients to create, start, and finish transactions, monitors the lifecycle and distribution of executed transactions, and is responsible for ensuring the ACID properties of executed transactions.

For ensuring transaction's ACID properties the transaction manager employs two other entities that are usually not visible from clients – the lock manager and log manager. A *lock manager* is responsible for transaction isolation and achieves it by locking. The lock manager locks data objects if they are manipulated by a transaction. To ensure transaction isolation, all locks are held until the transaction is committed. Each resource manager usually has its private lock manager that manages transaction-aware locking, i.e., locks are associated with transactions (e.g., this is the case with traditional relational databases). To achieve atomicity and consistency, the transaction manager orchestrates *recovery* in case of a TPS *failur*e. A failure could be a crash of one of the participating hosts, a hard disk failure, a network disconnection, a power failure, or a software fault. Recovery is based on ensuring durability of the committed transactions' effects and discarding the effects of transactions that were being executed at the time of the failure and thus will be aborted.

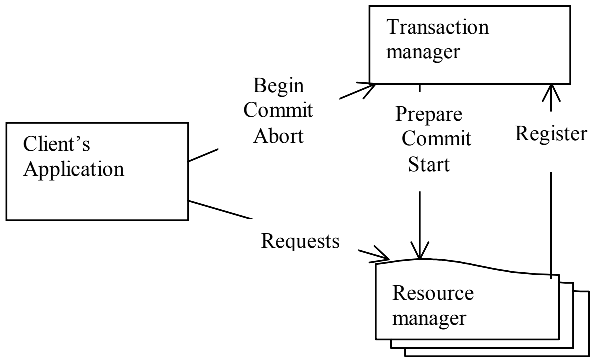

Figure 3: Basic Model of Transaction Processing System

Logging is the principal service that is used to support recovery. The *log manager* keeps track of every operation executed on behalf of a transaction. It writes the information needed for data recovery in case of transaction abort or TPS restart to a file called *log file* or simply *lo*g. It uses the following techniques for ensuring that the log is always in a consistent state: two copies of the log file are kept in persistent storage, the log is always written before data in a persistent storage is modified, all writes to the log are covered by appropriate locks and as part of the commit, the log is always written to the persistent storage (the *force-log-at-commit* rule). These techniques are usually combined with *checkpointin*g, which periodically writes the TPS state to a persistent storage to speed up the potential restart.

Since its parts are distributed over different nodes of the network, the TPS provides transaction-aware communication where connecting, authorization, and delivery of data requests take place. Every data request is executed on behalf of a particular transaction and is associated with the transaction identifier. We say that data requests are executed *in the context of a transaction* or simply in a *transaction contex*t. Several resource managers can be involved in a particular transaction. The transaction manager allows registering of particular resource managers with a transaction and manages transaction commit. Each resource manager executes its *local* transactions; each of them does not cross the resource manager's boundary. Since the resource manager is responsible for ensuring the ACID properties of its local transactions, it usually has its own private transaction manager, lock manager, and log manager. All participating transaction managers form a hierarchy, where every local transaction managed by the participating resource manager is associated with a *global* transaction managed by the topmost transaction manager, which is called the *root transaction manager* or *commit coordinato*r. A non-root transaction manager controls the local transactions executed on the corresponding resource manager, or it is responsible for coordinating transactions on the corresponding subtree of participating transaction managers is shown as shown in Figure 4.

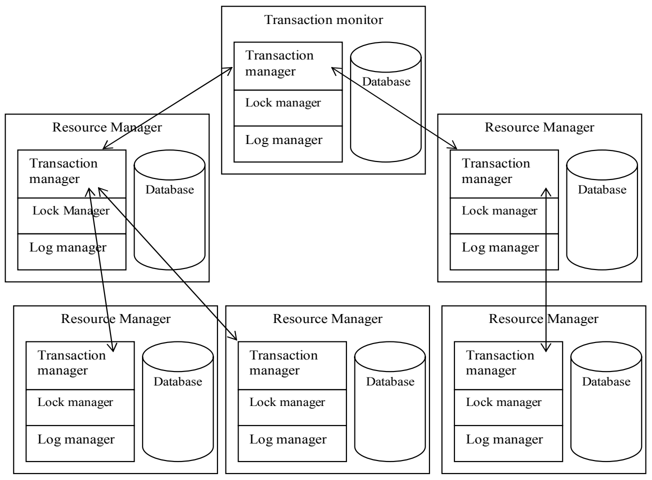

Figure 4: Architecture of Distributed TPS

In large distributed TPSs, a *transaction monitor* often plays the role of a central entity that controls global transactions. The transaction monitor (*TM*) is an application capable of integrating different heterogeneous TPSs and databases, and controlling several resources and terminals. The TM allows clients to initiate new transactions and to distribute transactions to several TPS on the network. They are also able to manage the ACID properties of executed transactions and, in particular, to perform recovery procedures. The TMs are designed to provide high reliability and performance. To achieve high throughput, they provide automatic load balancing [17], data request queuing, and other advanced features. If a TM is involved, clients never send their requests directly to the participating resource managers; instead, the transaction monitor mediates all the client's requests. The transaction manager of the TM often acts as the TPS commit coordinator.

The TPS is responsible for a negotiation protocol which guarantees that all effects of data requests executed on behalf of a transaction on registered resource managers are committed or aborted. In other words, all local transactions associated with a single global transaction are either committed or aborted. Usually, transaction managers support the *two-phase commit protocol.* In the first phase, the commit coordinator sends the *PREPARE* message to subordinate transaction managers. This is done recursively so that every transaction manager receives the *PREPARE* message. Each transaction manager votes *yes* or *no* indicating whether it is about to commit or abort. This is again provided in a hierarchical manner: a transaction manager coordinating its subtree's commit sends its vote message (vote for short) after it receives votes from all of its subordinate transaction managers. If all the transaction managers in the subtree are about to commit, then they vote *yes* and the subtree coordinator sends the *yes* vote to its parent commit coordinator. If any transaction manager in the subtree is about to abort, it sends *no* to the subtree coordinator, which then sends *no* to its parent coordinator. At the top level, if the commit coordinator receives *yes* from all of its subordinate transaction managers, it starts the second phase of the commit protocol by sending the *COMMIT* message to them. The message is then recursively sent to all the transaction managers and the transaction is committed. If the commit coordinator receives *no* from at least one of its subordinate transaction managers during the first phase of the commit protocol, it starts the second phase by sending the *ABORT* message to all of its subordinate transaction managers. The message is then propagated to all transaction managers in the hierarchy and the global transaction is aborted. The top-level transaction coordinator waits for messages acknowledging that all of the transaction managers have finished the second phase of the commit protocol.

Several optimizations of the two-phase commit protocol have been proposed in literature. For example, if a transaction is *read-only* (i.e., it does not provide any modifications of data objects), it can be committed in one phase. Advanced resource managers provide *heuristic decisions* on committing: a particular resource manager is able to heuristically commit or abort before the two-phase commit negotiation is completed. This can be efficient in situations when transaction managers have some advanced knowledge about the probability of commit or abort. If a particular transaction manager heuristically finishes a local transaction, and if his heuristic decision is wrong and does not correspond to the final vote of the global transaction, then the transaction manager has to provide an extra policy, which usually results in a human intervention. Several variants of the two-phase commit protocol is our next goal of this research that will support different communication topologies or to increase reliability in case of a commit coordinator or a participating transaction manager failure.

7. TRANSACTION MODEL

We have assumed that the open network environment is divided into network domains, regions (sub networks) and local sites of the clients as shown in Figure 5. The TPS are geographically distributed at different network domain, region and sites. There is a domain management server (DMS) in each network domain, which has information about all other DMS in the open network. One TPS running in a network domain considered as DMS. A transaction that is submitted to be performed over the open networks is called a global transaction. A global transaction is composed of a set of sub transactions. Each sub transaction may by its turn also contain sub transactions. The global transaction, therefore, has the form of a tree, called the transaction tree. The DMS of this tree is called the root transaction. The term transaction will be used hereafter to denote both the root transaction and sub transactions. Other common terms for hierarchical structures will also be used hereafter, such as client (leaf) transaction, parent transaction, etc.

The transaction running on the DMS is an open nested transaction. Each of the sub transactions of it can be either a flat ACID transaction or an open transaction. Open sub transactions of the DMS transaction have the same structure as the DMS transaction, thus applying the transaction structure recursively. Each of the flat transactions represents a client of the transaction tree.

Transaction running on the gateway corresponds to a combination of its sub transactions, forming a potentially complex control flow. The control flow of a transaction running on the gateway may include, for example, the specification of parallel and sequential execution of sub transactions, dynamic creation of sub transactions (instances) during the execution of a transaction and the definition of sets of alternative sub transactions (i.e., transactions that are equivalent, according to application semantics).

Each transaction has associated with it a set of input and output parameters, allowing a definition of data flow between transactions. Additionally, each transaction has a set of internal data which represents its private variables (its private state space).

The control flow of a transaction may be determined with the use of values of internal data and output parameters or outcome of previously executed transactions. The control flow is however, restricted in the basic transaction model so that for each transaction: (1) Open sub transactions can execute in parallel (2) The execution of flat sub transactions must be a sequence (3) No flat and open sub transactions can execute in parallel.

All transactions in this model are compensatable. Each transaction, with exception of the DMS transaction, has a corresponding compensating transaction. In case the effects of compensatable transaction must be cancelled after its commitment, its compensating transaction is executed. A compensating transaction cancels the effects of the compensated- for transaction according to application semantics. Compensation is performed in the reverse order of execution of the compensated - for transactions.

The compensating transaction of a local transaction of a client transaction is another flat transaction defined by the transaction specifier. The compensating transaction of an intermediary (i.e., transactions on the gateway) transaction corresponds to another open transaction that compensates the committed sub transactions of the compensated- for transaction. The compensating transaction for an intermediary transaction is defined automatically at runtime, depending on the sub transactions that have committed. Values for parameters of compensating transactions are defined by the application when the compensated- for transaction is committed or can be determined at the moment the compensating transaction executes.

DMS (root) also has information about all the regions in the network domain. DMS is responsible for maintaining uniqueness of names of regions, which are part of that network and helps to identify the region in which a transaction is present. Each DMS maintains a Domain Transaction Database (DTD), for information about the current location of all the transactions which were created in that domain or transited through it. Mobile transactions might have to split their computations into sets of operations, some of which operate on a MH while others on a FH. Frequent disconnection and mobility results in mobile transactions sharing their states and partial results violating the principle of atomicity and isolation which is traditional problem in existing transaction models. Mobile transactions require computations and communications to be supported by FH. Transaction execution may have to be migrated to a FH if disconnection is predicted in order to prevent the transaction from being aborted. The DMS behaves like a proxy and executes the transaction on behalf of the disconnected MH. The MH may either fully delegate authority to the DMS to commit or abort the transaction as it sees fit or may partially delegate authority, in which case the final decision to commit or abort the transaction would be made by the MH upon reconnection.

Each entry of DTD of the form (T_x, FD, r) represents that transaction T_x can be found in region r of the foreign network domain FD (foreign Network domain), or it has transited from that network domain or region r. For DTD and RTD (Region Transaction Database), the primary key is the transaction name T_x. With the help of these naming schemes we check the fault tolerance by maintaining the status report of mobile transaction which keeps the updated information of all the transactions. Transaction is migrated from one network domain to another through the DMS. During inter domain migration the transaction has to update location information in the DTD of the present domain and register in the DTD of the target network domain. Every region maintains information about all TPS that are part of that region. A TPS can be a member of an existing region or can start in a new region. In each region, a RTD is present at a TPS which runs at the gateway of a sub network. It contains location information about each mobile transaction that was created in that region or transited through it. This host acts, as the Transaction Name Server (TNS) [25], which manages the RTD. TPS is responsible for maintaining uniqueness of names of all transactions, created in that region. Generally a transaction name comprises of User Assigned Name, Birth Host, Region, and Network Domain. When a new transaction is created, the user assigns a name to it by registering in the RTD of its birth region. Each entry of RTD of the form (T_x, r, Nil) represents the region r where transaction T_x was found or transited through it. Similarly $\underline{(T_x, NIL, TPS)}$ represents transaction T_x, which exists in that region at TPS. For intra region migration, it has to update its location information in the RTD of that region. This is an Intra Region Location Update. During inter region migration, the transaction has to update the location information in the RTD of present region and register in the RTD of the target region, specifying the host in that region to which it is migrating.

DAD Tuple	**Meaning**	**RAD Tuple**	**Meaning**
(T_x, FD, r)	Transaction T_x is in region r of	(T_x,r, NIL)	Transaction T_x is in present network

DAD Tuple	Meaning	RAD Tuple	Meaning
	the foreign network domain (FD)		domain and in region r
		(T_x NIL,TPS)	Transaction is in present network domain and region at site TPS

Table 1. DTD and RTD Tuples

Any location protocol for mobile transactions deals with three aspects: **name binding**, **migration** and **location,** each related to a particular phase in the transaction's lifetime. We have defined four atomic operations which are incorporated on **DTD** and **RTD**

1. **bind** operation is performed when a name is assigned to a newly created transaction, whose birth location is also stored. This operation causes the insertion of a new tuple in the database. As the transaction name has to be unique, this operation fails if a tuple with the same name already exists in the database.
2. **newloc** operation is performed when the transaction changes its location, by migrating to a new one. This operation updates the tuple already present in the database.
3. **find** operation is performed when a transaction has to be located in order to interact with it. For a given transaction name, this operation returns the current location of the transaction.
4. **unbind** operation is performed when a transaction name is no longer used (i.e., the transaction has been disposed off). This operation causes the deletion of the relative tuple from the database.

Since locating the transaction requires following a long path before reaching it. It follows a part of the link the transaction has left on the registers of the visited region and the network and parent domains. The updating operations performed during the *migration* phase are designed in order to shorten this path, thus increasing interaction efficiency and reducing the overhead. The steps for locating a target transaction T_i are as follows:

1. Extract birth network domain and birth region name from T_i.

   Domain_name $\leftarrow$ Birth_Domain_Name;
   Region_Name $\leftarrow$ Birth_Region_Name;
2. Contact relative DMS.
3. If query to DMS results in a tuple (T_i, FD_i, R_i) {target transaction is not in that domain}

   Domain_Name $\leftarrow$ FD_i;

   Region_Name $\leftarrow$ R_i;

   Get the domain name from the tuple and go to Step 2.
4. Else contact relative RTD //Transaction exists in the given Region_Name
5. Get the query result tuple (T_i, R_i, TPS_i)

   Region_Name $\leftarrow$ R_i;

   TPS_Name $\leftarrow$ TPS_i;
6. If R_i is *Nil* target transaction is located at TPS_i, else go to Step 4.

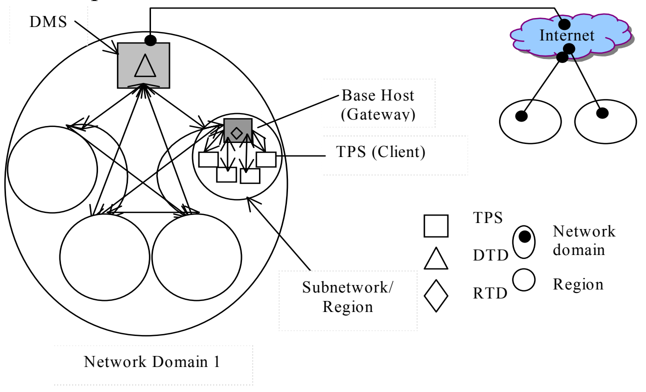

Figure 5: A Hierarchical Mobile Transaction Model for Open Network.

It is up to the binding and migration phases to maintain consistency of location information in the databases in such a way as to always allow transaction finding (unless a system or network crash occurs). When the network domain, in which the transaction is present, is found, the DTD is locked. Similarly the RTD is locked when the region is traced. The lock is reset only if the transaction does not reside in that region. It should be noted that keeping the RTD locked prevents the transaction from further migration so that communication with it is possible. This is required when direct or synchronous transaction - transaction communication is needed. For asynchronous transaction - transaction communication a message is dropped in the mailbox at the gateway/DMS and the transaction receives this message when it wants. Other possible case is drop and delayed transaction- transaction direct communication. In this technique transaction multicast message to all the gateways of a network domain and when it finds acknowledgement that transaction is found in particular region. The transaction waits for the message-receiving transaction to contact this transaction for making the dialogue.

Each transaction is either vital or non-vital. A vital transaction is a transaction the failure of which determines immediately the failure of its parent transaction. A failure of a non-vital transaction does not have direct effects on the outcome of its parent transaction.

Each client transaction is restricted to be executed entirely at the same TPS, i.e., only service components at the same TPS are accessed as part of a client transaction. The control flow of a client transaction represents a combination of accesses to services at that TPS. A compensating transaction for a client

transaction is considered to be executed at the same TPS where the compensated- for transaction executed.
The general recovery semantics of the basic transaction model is as follows. In the occurrences of failures the recovery process of a transaction tries to perform forward recovery. A recovery process is performed which resets the execution to a consistent state and the transaction continues to be executed from that state on, trying to achieve a successful termination state. Backward recovery i.e., the cancellation of the effects of a transaction, however, may also occur. Backward recovery is performed when a vital transaction aborts. In this case the parent transaction of the vital transaction will be backward recovery.

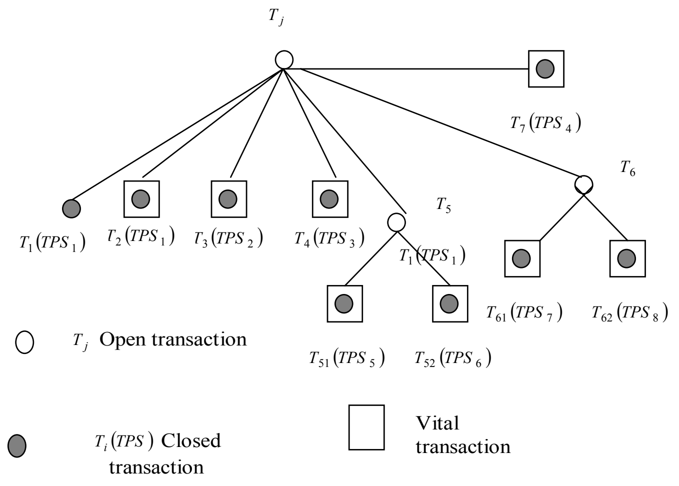

Figure 6: Flow of Transaction based on the Transaction model shown in Figure 5.

Due to the behavior of the fault tolerance mechanism, upon which this transaction model is based, partial backward recovery may also occur. In this case some of the already committed sub transactions of an open transaction are compensated as a form of back tracking the execution to a previous consistent state. Forward execution of the transaction is then performed from that state on. When the copy of the transaction at a TPS cancels the effects it produced after having stored the checkpoint. The basic transaction model enforces semantic atomicity.
Figure 6 shows an example basic transaction. In this transaction, the DMS transaction (T_j) has 7-sub transactions, denoted T_1 to T_7. Transactions T_1, T_2, T_3, T_4, and T_7 are closed. Transactions T_5 and T_6 are open. Transactions T_1 and T_2 should be executed, respectively, at TPS_1. Transaction T_3, T_4, and T_7 should be executed, respectively, at TPSs TPS_2, TPS_3 and TPS_4. The open transaction T_5 has two closed sub transactions, T_{51} and T_{52}, to be executed, respectively, at TPS_5 and TPS_6. Similarly open transaction T_6 has two closed sub transactions T_{61} and T_{62}, to be executed, respectively, at TPS_7 and TPS_8. Transactions T_2, T_3, T_4, T_7 and all the sub transactions of T_5 and T_6 are vital. Each transaction is either vital or non-vital. A vital transaction is a transaction the failure of which determines immediately the failure of its parent transaction. If any of them fails, its parent transaction must be backward recovered. A failure of a non-vital transaction does not have direct effects on the outcome of its parent transaction.
Figure 7 shows the control flow defined for the open transaction T_j. As shown in Figure 7, T_2 will be executed if T_1 succeeds. Transaction T_3 is executed if T_2 succeeds. Transaction T_4 is executed if T_1 fails. Transaction T_5 and T_6 are executed in parallel, after either T_3 or T_4 succeeds. Transaction T_7 will execute after T_5 and T_6 terminate. Similar definitions of control flow are supposed to exist for open transactions T_5 and T_6.

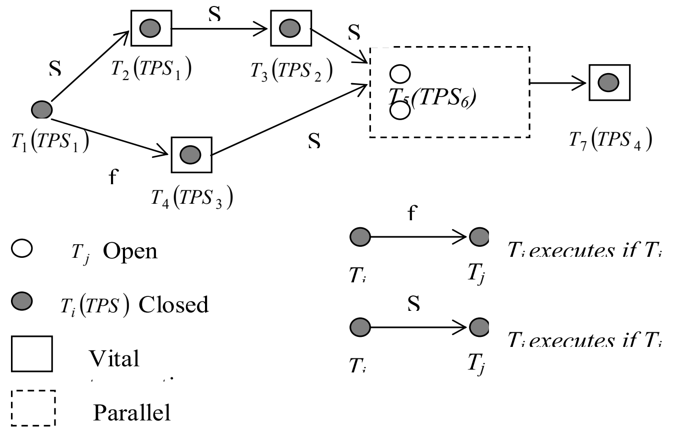

Figure 7: A Representation of control flow for root transaction of Figure 6

8. IMPLEMENTATION

To study the performance of the model suggested in section 6 we have implemented it on 10/100/1000 Mbps switched LAN that connects 850 workstations and personal computers, and is used by about 500 researchers and students. Machines are grouped into eight different networks with their own servers and servers of each network are connected to the main server of the institute. For each network there are 100 nodes which are running TPS, three mobile stations running TPS (DMS). These DMS are running mobile codes for finding the status of the different sub-transactions in different networks whenever a failure is arise. Mobile codes are implemented on PMADE [20]. We have implemented the transaction for computing the prime numbers (between 1 and 9999999) on a cluster of PCs (P-4, 3 GHz machines) using PMADE and j2sdk1.5.1

8.1 Performance Study

Figure 8 shows the system throughput of two approaches (developed and Kangaroo Model [9]). The throughput of the developed scheme is close to the Kangaroo Model in all case. As the developed model is implemented in Java, the high execution overhead of Java program results in the lower throughput when number of the TPSs very high. The real overhead generated due to DMS (root) controller of the sub-transactions which monitoring the status of them. The DMS launches a mobile code in case of a failure arise on any TPS for recovering the failed sub-transaction.

Figure 9 compares the system throughputs of the developed system with kangaroo model when temporary faults arising at different servers randomly. The result shows that the developed scheme can obviously improve the system throughput when increasing the number of TPS (servers). In the latter case, the processing capacities of the TPSs are wasted and no improvement.

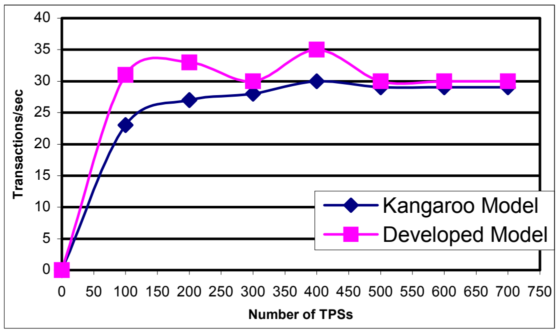

Figure 8: System throughput

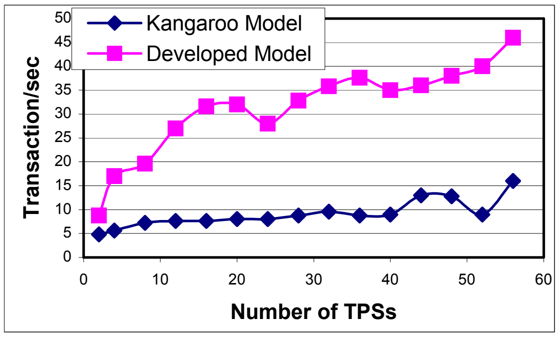

Figure 9: System throughputs of the developed scheme and the Kangaroo model when temporary faults are arising on different TPSs

8.2 Comparison With Existing System

Because in normal kangaroo transaction model (KT), three potential problems arise:

1. Resource blocking for other relatively smaller transactions initiated by the same user while main long lived transaction (LLT) is waiting for inputs. This is because most of the available commercial DBMS packages use conventional two-phase locking protocol [27].
2. Even if resource blocking doesn't occur due to usage of independent database resources by transactions, separate kangaroo transactions has to be initiated in each case the user initiates even a small transaction while the LLT is running. This leads to wastage of BS server resources.
3. Failure of a global transaction in a Joey in compensating mode results in abortion of the entire KT. Thus even if some transactions are there which are short and not involved in the failure, they will get aborted unnecessarily.

Transaction Model	**Atomicity**	**Consistency**	**Isolation**	**Durability**	***Execute In***
Kangaroo	May be	No	No	No	*Fixed Network*
Developed	*Yes*	*Yes*	*Yes*	*Yes*	*Both Fixed and Mobile Ad hoc Networks*

Table 2: Comparison between Developed Model and Kangaroo Transaction Model

9. RELATED WORKS

This section will review the transaction-processing concept and discuss on mobile transaction. Location and time of MH are the two major factors that effect on mobile transaction processing. This section outlines three mobile transaction models, which focus on mobility of MH.

Moflex transaction model [8] allows to model mobile transaction with extra information such as location, time and the precondition of mobile transactions. The sub-transaction *T* of mobile transaction *M* can be executable only when its external precondition predicated is satisfied. Moflex takes into account which sub-transactions are location-dependent.

Pre-write transaction model [4] allows a transaction on a mobile host to submit a pre-commit state and the rest of the transaction can be carried out at the fixed or other mobile hosts at later time. The main point is making all the updated data items visible to other transactions. This model can be use to support mobile hosts which have little power for processing data. Pre-commit transaction model eases the locking on data record and avoid longer time blocking other transactions. However it is not carefully taking into account the risk of frequent disconnecting and higher error rates of wireless data transmission.

Kangaroo transaction model [9] is developed beside the existing multi-database environment. Kangaroo mobile transaction does not start and end at the same host. In this model, mobile transaction hops through stationary hosts in wired network. The whole transaction and related information are pushing forward to the final committed host. Kangaroo model is supported by the autonomy of local DBMS. Kangaroo is one model that captures the movement nature of mobile unit. Recovery from long term failures of the nodes from where a transaction is being controlled and mobility of the control flow of a transaction execution were also considered in the development of two transaction models, respectively, in the transaction model of ConTracts[11] and in migrating transactions[12]. In the ConTracts, if the node from where a ConTract is being executed fails, it can be re-instantiated at another node. A ConTract, however, does not move during its execution. In migrating transactions, the flow of control of a transaction migrates in a distributed environment. Executing transactions with mobile codes extends the notion, providing more flexibility for the distribution of code and for the movement of the transaction control flow in the environment.

In [13] the fault tolerance protocol and the transaction model presented here are described in details. In this model aspects of the presented approach are further discussed, such as: extensions to the basic transaction model; replication policies considering the availability properties of agencies; how autonomy of system is supported by the model; among others.

In [18] a concept is presented for executing open and closed nested transactions with multiple mobile agents. The paper, however does not consider long-term failures. In [19] a model for executing transactions with a single mobile agent is presented. The transaction model presented supports compenstable and non- compenstable transactions and the specification of so-called ACID groups. An ACID group is a combination of sub transactions that is executed isolated from other parts of the same transaction and from other parts of the same transaction and from other agent-based transactions. The model supports that ACID groups or the set of non-compenstable transactions span more than a single agency. In this paper the execution of distributed transactions can be based on more than a single mobile agent. Additionally, it is not allowed here that isolated parts of an agent-based transaction span more than one agency, in order to facilitate recovery from long term failures.

The developed hierarchical mobile transaction model is fault tolerance in case of temporary failures arise on the transaction execution servers and gives better performance than Kangaroo model.

10. CONCLUSION AND FUTURE WORKS

In this paper we have presented a hierarchical mobile transaction system. This transaction model is based on mobile codes that take into considerations properties and requirements of open networks and their applications. The model represents a concept that integrates the mobility of the codes with the execution of control flows with transaction semantics. This transaction can be used as an approach for providing reliability and correctness of distributed activity in the open networks that provides the benefits of mobile codes. The resulting concept exhibits important features that should be supported by an underlying infrastructure to fulfill requirements of applications running in open networks.

The effectiveness of the applicability of mobile codes to open environments is, however, subjected to or influenced by the development of appropriate solutions to a set of issues. The main set of such issues realties to what can be called controllability of mobile code based activities. Other aspects are security, accounting and testing. The scope of applicability of mobile codes will be dependent on the achievements reached to these issues. The described model represents a step towards developing controllable mobile code based activities. This model is currently being extended to incorporate more functionality and to decrease some of the implied costs.

The mobile computing environment can support MHs to perform mobile transaction. Users can easily manipulate information despite of their location and what mobile devices they have. However, the disadvantage of this environment is that it cannot provide flexible way to exchange data between MH. One BH responds for supporting all MH in its region, this can cause a bottleneck when there are many MH in the same region and single failure mode if this BH fails. Current mobile transaction models are based on existing database systems. The models along with the characteristics of mobile environment help to analyze the requirement of mobile applications. The challenge is that when every host in mobile environment is MH. Different variants of the two-phase commit protocol is our next goal of this research that will support different communication topologies or to increase reliability in case of a commit coordinator or a participating transaction manager failure

REFERENCES

[1] Jim Gray, Andreas Reuter (1993), Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publishers, Inc.

[2] George Coulouris, Jean Dollimore, Tim Kindberg (2001), Distributed Systems: Concepts and Design. Addison-Wesley.

[3] V. K. Murthy (2001), Seamless Mobile Transaction Processing: Models, Protocols and Software Tools, in Proceedings of the Eight International Conference on Parallel and Distributed Systems (ICPADS 2001), 26-29 June 2001, KyongJu City, Korea, pp. 147-156.

[4] Madria, S.K., Bhargava, B (1998), A Transaction Model for Mobile Computing, in Proceedings of the International Conference on Database Engineering and Applications Symposium (IDEAS'98), Cardiff, Wales, U.K., July 08 - 10, 1998, pp. 92-102.

[5] M. Tamer Ozsu, Patrick Valduriez (1999), Principles of Distributed Database Systems Prentice Hall.

[6] George H. Forman, John Zahorjan (1994), The Challenges of Mobile Computing, 27(4): 38 – 47, April.

[7] Karen Furst, William W. Lang and Daniel E. Nolle, See Furst, Karen, William W. Lang, and Daniel E. Nolle (1998), Technological Innovation in Banking and Payments: Industry Trends and Implications for Banks, Quarterly Journal, Office of the Comptroller of the Currency, Vol. 17, No. 3, pp. 28, Sept.

[8] Kyong-I Ku; Yoo-Sung Kim (2000), Moflex transaction model for mobile heterogeneous multidatabase systems, in Proceedings of the 10th International Workshop on Research Issues in Data Engineering (RIDE 2000), San Diego, CA, USA. Feb.28-29, 2000, pp. 39 –45.

[9] Margaret H. Dunham, Abdelsalam Helal, Santosh Balakrishnan (1997), A Mobile Transaction Model that Captures Both the Data and Movement behavior, Mobile Networks and Applications, 2, pp. 149–162.

Continued on page no. 84