Mobile Transactions*

M. Brahmaji Rao (Y111117)
Ch.Srinivas Rao (Y111126)
P. Narender (Y111119)

Instructor: Dr. R. K. Ghosh

Abstract

Mobile applications require transaction like properties. In this project we implemented two models: Kangaroo Transaction model and Team transaction model and compared the pros and cons of the two models. Kangaroo transaction model is built on the concepts of split transactions and global transactions in a multidatabase environment. Kangaroo Transactions incorporates the property that transactions in a mobile computing system hop from one base station to another as the mobile unit moves through cells. Team transaction Model is proposed for ad-hoc grouping of mobile hosts. This model has a three level control hierarchy just like any sport team namely the bench, the coordinator on the field and the individual team members. Team transaction model uses extensive recovery mechanism which is very much needed in mobile transaction environments. Finally we compared the two models.

1 Introduction

Recent advances in hardware technologies such as portable computers and wireless communication networks have led to the emergence of mobile computing systems. Examples of mobile systems that exist today include travelers carrying portable computers that use cellular telephony for communications, and transporting trains and airplanes that communicate location data with supervising systems. Access by users with mobile computers to data in the fixed network will involve transactions. However a transaction definition in this environment varies from that within a centralized database or even a distributed environment. In addition, limitations of the wireless and nomadic environment place a new challenge to implementing efficient transaction processing using classical transaction models. Disconnection is the major obstacle.

The ACID properties in mobile environment are a bit relaxed in nature. There are several existing transaction processing models. We first described the Kangaroo transaction

*This project is done as part of the course CS634: Mobile Computing under the guidance of Dr. R. K. Ghosh

model, then Team Transaction model and lastly the comparisons. The recovery scheme in Kangaroo transactions relies on compensating transactions where as the scheme in team transaction makes use of message logging concepts for recovery and rollback.

2 Mobile database environment

The environment consists of stationary and mobile components. A Mobile Unit is a mobile computer which is capable of connecting to the fixed network via a wireless link. A Fixed Host is a computer in the fixed network which is not capable of connecting to a mobile unit. A Base Station is capable of connecting with a mobile unit and is equipped with a wireless interface. They are also known as Mobile Support Stations. Base stations, therefore, act as an interface between mobile computers and stationed computers. The wireless interface in the base stations typically uses wireless cellular networks.

The reference model consists of three layers: the source system, the data access agent, and the mobile transaction. The Source System represents a collection of registered systems that offer information services to mobile users. Data in the source system(s) is accessed by the mobile transaction through the Data Access Agent (DAA). Each base station hosts a DAA. When DAA receives a transaction request from a mobile user (we call this a mobile transaction), the DAA forwards it to the specific base stations or fixed hosts which contain the needed data and source system component. When the mobile user is handed over to another base station, the DAA at the new station receives transaction information from the old base station. The mobile user accesses data and information by issuing transactions. Mobile Transaction is defined as the basic unit of computation in the mobile environment. It is identified by the collection of sites (base stations) it hops through. These sites are not known until the transaction completes its execution. A major function performed by the DAA is management of the mobile transaction. This component of the DAA is called the Mobile Transaction Manager (MTM).

3 Kangaroo Transactions

This model [1] is built on traditional transactions which are a sequence of operations executed under the control of one DBMS. Figure 1(a) shows the basic structure. Here three operations (op11, op12, and op13) are performed as part of the transaction. LT is used to identify the traditional transaction as it is executed as a Local Transaction to some DBMS. The operations performed are the normal read, write, begin transaction, abort transaction, and commit transaction. The first operation (op11) must be a begin transaction while the last (op13) must be either a commit or abort. We actually consider two types of global transactions. The limited view of global transactions is shown in figure 1(b). Notice that the Global Transaction root (GT) is composed of subtransactions which can be viewed as Local Transaction (LT) to some existing DBMS. The local transactions are often called subtransaction (or Global SubTransaction, GST) of the GT. Each of these can in turn be viewed as consisting of a sequence of operations. Figure 1(c) takes the more general view of global transactions. In this case the subtransactions may themselves be

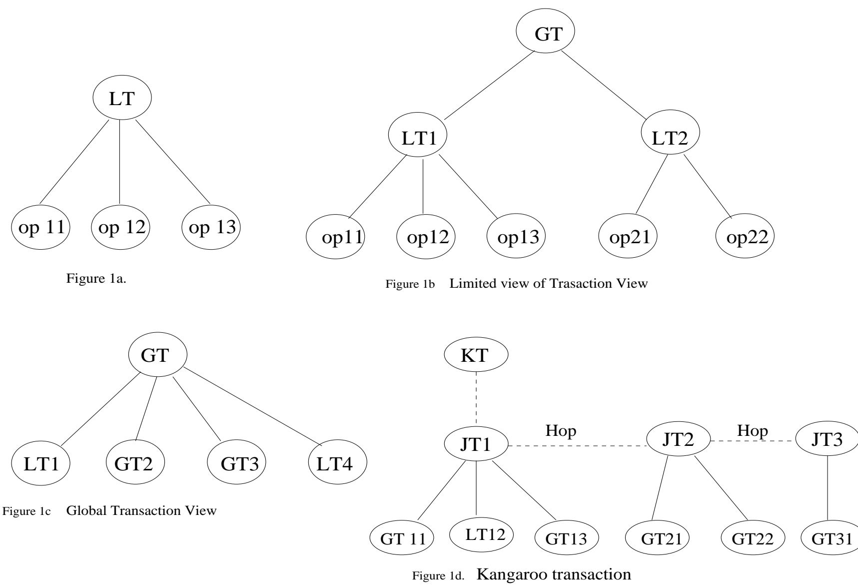

Figure 1: Kangaroo Transaction

global transactions to another multidatabase system. So we would have (for this figure) operations underneath LT1 and LT4. Underneath GT2 and GT3 would be other LTs and GTs. This view of global transactions gives a recursive definition based on the limiting bottom view of local transactions.

3.1 *Introducing Kangaroo Transactions*

Global transactions serve as the basis upon which mobile transactions are defined. Global transactions alone, however, do not capture the "hopping" nature of mobile transactions. Based on the hopping property, we call our model of mobile transactions Kangaroo Transactions

Figure 1(d) shows the basic structure of a Kangaroo Transaction. When a transaction request is made by a mobile unit the DAA at the associated base station creates a mobile transaction to realize this request. A Kangaroo Transaction ID (KTID) is created to identify the transaction. KTID is defined as follows:

KTID = Base Station ID + Sequence Number,

where the base station ID is unique, the sequence number is unique at a base station, and + is a string concatenation operation. Each subtransaction represents the unit of execution at one base station and is called a Joey Transaction (JT). The origination base station initially creates a JT for its execution. The only difference between a JT and a GT is that the JT is part of a KT and that it must be coordinated by a DAA at some base station site. When the mobile unit hops from one cell to another, the control of the KT changes to a new DAA at another base station. The DAA at the new base station site creates a new JT (as part of the handoff process). It is assumed that JTs are simply assigned identification numbers in sequence. Thus a Joey Transaction ID (JTID) consists of the KTID + Sequence Number. This creation of a new JT is accomplished by a split

operation. The old JT is thus committed independently of the new JT. In figure 1(d), JT1 is committed independently from JT2 and JT3. At any time, however, the failure of a JT may cause the entire KT to be undone. This is only accomplished by compensating any previously completed JTs as the autonomy of the local DBMSs must be assured.

To manage the KT execution and recovery, a doubly linked list is maintained between the base station sites involved in executing a Kangaroo Transaction. Control information about a JT is identified by its JTID. To complete a partially completed KT, this linked list is traversed in a forward manner starting at the originating base station site. Thus to restart an interrupted transaction, the user must be able to provide the starting site (base station) for the transaction. To undo a KT, the linked list is traversed in a backward manner starting at the current JT base station site.

There are two different processing modes for Kangaroo Transactions: Compensating Mode and Split Mode. When a KT executes under the Compensating mode, the failure of any JT causes the current JT and any preceding or following JTs to be undone. Previously committed JTs will have to be compensated for. Operating in this mode requires that the user (or source system) provide information needed to create compensating transactions. This includes information that the JT is compensatable in the first place. Deciding whether a JT is compensatable or not is a difficult problem. Not only does the JT itself need to be compensatable, but the source system should also be able to guarantee the successful commitment of the compensating transaction. The split mode is the default mode. In this mode, when a JT fails no new global or local transactions are requested as part of the KT. However, the decision as to commit or abort currently executing ones is, of course, left up to the component DBMSs. Previously committed JTs will not be compensated for. Neither the Compensating nor Split modes guarantees serializability of the kangaroo transactions. Although Compensating mode ensures atomicity, isolation may be violated (thus violating the ACID principle) because locks are obtained and released at the local transaction level. With the Compensating mode, however, Joey subtransactions are serializable.

Figure 2 shows the relationship between movement of a mobile unit between cells and the corresponding Kangaroo Transaction. Here we assume that when the transaction is started, the mobile unit is in Cell 1 which is associated with Base Station 1. At this time, the DAA at this Base Station created a new Kangaroo Transaction and immediately created a Joey Transaction. When the mobile unit moves to Cell 2, a handoff is performed. As part of this handoff, the KT is split into two transactions. The first part of this transaction is the subtransactions under JT1, the remainder (at this time) will be part of JT2. Similarly, when the mobile unit moves into Cell 3 and Cell 4, handoffs occur and new Joey Transactions are created via a split operation. Note that this process is dynamic. A new Joey is created only when a hop between cells occurs: no hop no Joey. The same transaction requested at two different times could have different structures.

3.2 *Mobile transaction Manager data structures*

The functions of the MTM are those related to managing a mobile transaction. The primary data structure at each site which is used to do this is the transaction status table (see Figure 3).

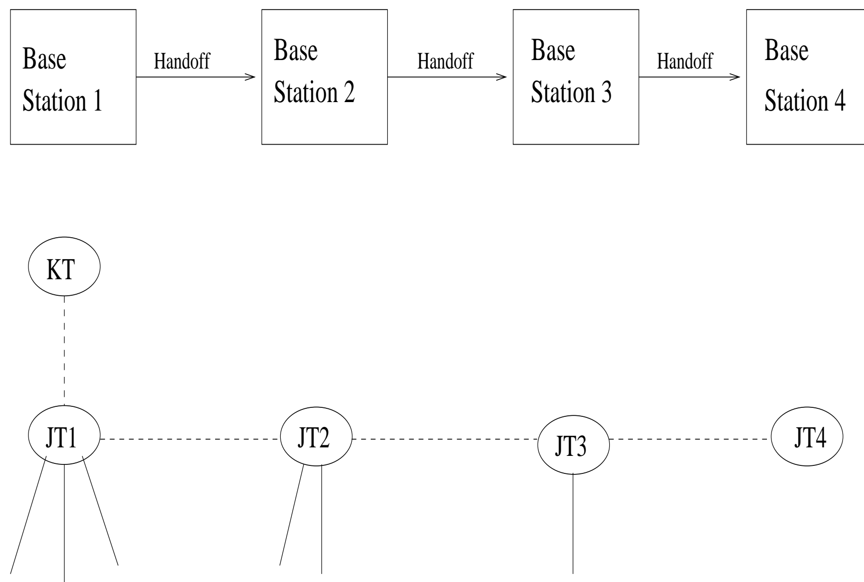

Figure 2: Hopping from Base Station to Base Station

Each base station maintains a local log into which the MTM writes records needed for recovery purposes. Unlike DBMS and GDBS systems, this log contains no records dealing with recovering (undo or redo) database updates. In the implementation it is assumed that SampleData_1.db, SampleData_2.db, SampleData_3.db, SampleData_4.db are the data files at Site1, Site2, Site3 and Site4 respectively. The Sample transaction file considered is trans.jb. It contains all the arithmetic operations since compensating(simply inverse)operations can be easily defined.

3.3 *Kangaroo Transaction processing*

The flow of control of processing Kangaroo Transactions by the MTM can be described as follows:

1. When a mobile unit issues a Kangaroo Transaction, the corresponding DAA passes the transaction to its MTM to generate a unique identifier (KTID) and creates an entry in the transaction status table. The MTM also creates the first Joey Transaction to execute locally in its communication cell.

2. The creation of a Joey Transaction (be it the first or otherwise) is also done by the MTM and involves generating a unique JTID and creating an entry in the transaction status table. The count of number of active Joeys in the KT status table entry is incremented by 1. Finally a JT entry is written into the transaction status table.

3. The Kangaroo Transaction is executed in the context of Joey Transactions. For each JT, translation is made to map the KT operations into specific source system

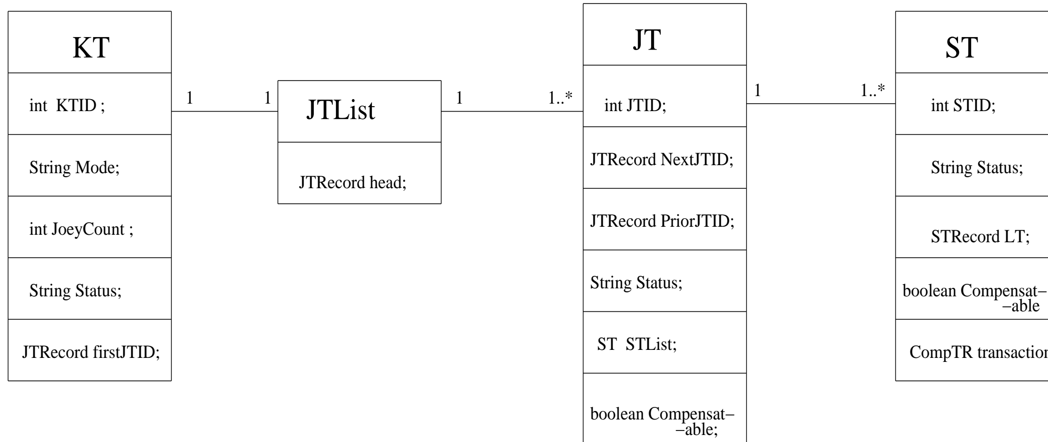

Figure 3: KT transaction status table

global and local transactions.The first part of trans.jb file temp1_1.jb is applied on SampleData_1.db file.

4. When network-level handoff occurs, the DAA is immediately notified so that it responds by initiating a transaction-level handoff protocol. When this takes place, the DAA starts executing a split operation at the origination site. As part of split the transaction file temp1_2.jb is divided into temp2_1.jb and temp2_2.jb.

5. The destination base station then initiates the other part of the split operation. A new Joey Transaction is created with the tentative content being the rest of the entire Kangaroo Transaction. In addition, a KT entry is placed in the destination's status table. Now temp2_1.jb is applied on SampleData_2.db.

6. When the DAA is notified (by the DBMS or GDMS) that a subtransaction has been committed, and if the mode is Split, then the ST entry in the transaction status table is removed. If the KT mode is Compensating, this entry remains in case the KT is later aborted and the compensating transaction must be executed. The STList in the status table is updated to reflect the fact that this subtransaction has committed. If there are no active subtransactions for this Joey, then this Joey transaction is committed. Finally, the count for number of active Joeys is decremented by 1. Note that the KT status table entry for the last active Base Station contains the current values for Status and Joey count. If the Joey count is 0 and the status of the KT is Committing, then the KT is committed.

7. When the mobile user indicates that the transaction has ended, the status entry for the KT is changed to Committing. At this time, if the active Joey count is 0 the KT is committed.

4 Sequence Diagram

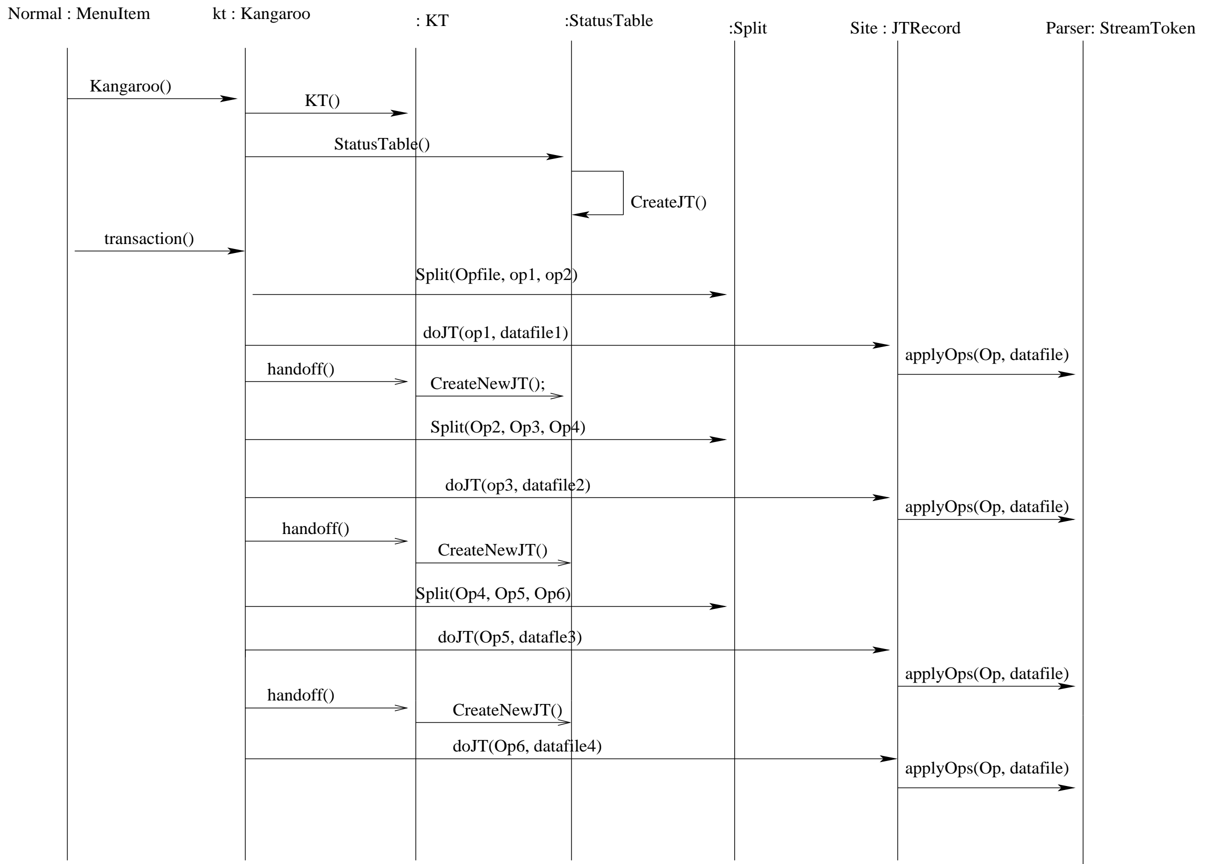

Figure 4: Normal Kangaroo Transaction Processing

5 The Team Transaction Model

Team transaction model [2] represents the idea of executing transactions in a single cell having mobile hosts. A subset of these hosts will form a team to do a transaction in a collaborative manner. We call the Mobile Support Station (MMS) in which all these mobile hosts lie as *bench* in the team transaction terminology. Usually a number of mobile hosts (MH) are found under a cell. Thus a number of hosts can connect to a single *bench*. A *bench* picks one uncommitted transaction and assigns it to one of the mobile hosts in its cell. This action is performed until no more uncommitted transactions can be found or no free host is available. The MH which gets a transaction from the *bench* is called a *coordinator* for that transaction.

The *coordinator* splits the transaction into subtransactions to form player transactions, and assigns these sub transactions to other willing MHs. Since all node are within the same cell, a call setup between them is inexpensive. While decomposing a transaction the *coordinator* takes into account the dependencies among the sub transcations.

The MH running the coordinator for a transaction and the MHs executing the player transaction of that coordinator constitute a team for that transaction, and hence the name *team transaction*. The player MHs perform the transaction assigned to them by the *coordinator* and communicate with *coordinator* through messsages. Finally, *commit* message is sent to the *coordinator*. The *coordinator* also exchanges the same type of message with the *bench*.

As discussed above the main tasks performed at different levels of control for the completion of a transaction, and involves following main actors:

1. Bench
2. Coordinator
3. Mobile host
4. Recovery System

5.1 Bench

The *bench* has a list of transactions that are to be executed, and the list of MHs which are active under its cell. The *bench* picks up a transaction and assigns it to one free mobile host. This MH is referred to as the *coordinator* of that transaction. Each transaction is represented by a unique number `TTID`, and each mobile host by an identifier called `MHID`. The *coordinator* id is known as `CID`. As part of the strong recovery strategy used for this transaction model, the *bench* maintains a data structure called *action buffer*. The *bench* receives messages from the *coordinator* and performs message logging in the *action buffer*. The message logging is useful when recovery is needed. As each `TTID` is given to a `CID`, the *bench* maintains a (`TTID-CID`) list. When a message comes from a *coordinator* it checks `TTID`, originator ID, etc., in the message for the purpose of identifying, and storing the message at the appropriate position in the action buffer.

5.1.1 Action buffer

The *action buffer* is organized in the form of multilists. For each `TTID` one entry is maintained. There may be a list of `TTID`s and each `TTID` contains a list of `PTID`s, the player transaction IDs assigned by the *coordinator*. The *coordinator* sends the message informing which `PTID`s were assigned for a particular `TTID`. For each `PTID` the *action buffer* contains a list of messages sent by the `MHID` having the `PTID` assigned by the `CID` of corresponding `TTID`. So when a message is received at the bench then it checks the `TTID`, `PTID` in that `TTID` list and stores the message in the corresponding `PTID` list. When a *commit* is reached at the bench it commits that `TTID`, i.e., the entries in the corresponding action buffer are made permanent. Until the time *commit* has not reached, the messages in the log are only temporary. Table 1 describes various type of messages that may be received by *bench* and the corresponding actions that must be performed at the *bench* in response.

Message Type	Action by the Bench
COMMIT	Make the entries in *action buffer* permanant.
SPLIT DELEGATE	Choose another *coordinator* for that TTID
ROLLBACK	Remove entries in the *action buffer* asked by *coordinator*
DATA	Log message in *action buffer*

Table 1: Messages received by *the bench* and the corresponding action.

5.2 Coordinator

The main task of *coordinator* is to decompose a transaction with a given `TTID` into a number of player transactions, thus, generating a number of `PTID`s. The *coordinator* has to also maintain the dependency among the `PTID`s. Depending on the dependency order the `PTID`s are executed one by one and if no dependency means the `PTID`s can be executed parallely. Each `PTID` is given to a free `MHID`. The *coordinator* can also be a player for the transaction it is executing if no free `MHID` is available. Similarly, the same node can also be a player for many other `TTID`s. After assigning the work to each player (`MHID`) *coordinator* expects some messages from its players. It performs necessary actions depending on the type of message it receives. For purpose of processing the received information `CID` needs to store some data structures.

1. It should have `PTID` list for the given `TTID`
2. `PTID`-commit list for the purpose of checking which `PTID`s have committed

When a player sends a message to the coordinator it checks the type of message. Depending on the message type the `PTID`-commit list is modified and if all the players have given *commit* then final commit is passed to the bench for that `TTID`. This is the scenario when `MHID`s resides in the same cell upto the completion of the job assigned to it. Since this transaction model does not restrict the mobility of mobile hosts, there may be situation when MH leaves the cell without committing its job.

5.2.1 Timeout for each player

The *coordinator* maintains a timer for each mobile host for which it assigned a `PTID`. If this times out then the corresponding `PTID` is checked for commit and if that is not the case, then the same task is assigned to another `MHID` in the cell or the coordinator itself does the unfinished job. This case solves the problem of a mobile host crash.

5.2.2 Split-Delegate message by player:

When the mobile host is leaves the present cell then it sends *Split-Delegate* message. Then the coordinator assigns the work to another player. No message from the MHID after it had send a Split-Delegate message is accepted.

Table 2 gives a summary of the actions performed by the *coordinator* upon receiving various messages from the players.

Message Type	**Action at CID**	**Message to bench**
DELEGATE	Check if all PTIDs are finished	if YES send COMMIT message
DATA	Log the message ID	Send the message
SPLIT DELEGATE	Choose another MH and give the remaining work	Send ROLLBACK message with message IDs logged for uncommitted PTID

Table 2: Messages received by *the coordinator* and the corresponding action.

5.3 Mobile Host

The work of mobile host is to execute the work given by the *coordinator* and sending the information to the *coordinator* through messages. Due to the message overheads the host does not send any message until some useful data is generated. The structure of message is provided in Table 3.

Message ID	ID for the message generated, each message will be having distinct ID.
MHID	Identity of MH from which message is originated.
CID	Coordinator ID for the MHID.
PTID	Assigned PTID for the MH.
Type	Type of message: DATA, COMMIT, DELEGATE, SPLIT-DELEGATE
Data	It is the iformation field.

Table 3: Messages structure.

A normal message is marked as DATA. When the subtransaction is completed at the MH it sends a message of the type DELEGATE. If the host leaves the current cell without completing the work then it sends a message of the type SPLIT-DELEGATE.

6 Recovery

Since the hosts are mobile, they may go out of cell any time and also there are the chances of crashing. To handle these situations team transaction model maintains the stable storage log at the *bench* referred to as *action buffer*. It reflects the present state of each transaction. This model can guarantee recovery at the mobile host level, the coordinator level and also at the bench.

6.1 Recovery at the mobile host

As the host may be busy with the work assigned to it, and it may send wrong data to the *coordinator*. In such situtation, it has to *undo* what it has *done*. This is known as local rollback and generally happens with less probability. In this case no message is sent or received. It is done by the mobile host itself without the intervention from the *bench* or the *coordinator*.

6.2 Recovery at Coordinator

The mobile hosts to which `PTID`s are assigned may crash or migrates to a new cell without completing the task. Then the work done by them remain unfinished. The unfinished work should be removed and must be assigned to another players. As the data is stored at the *bench*, the *coordinator* should give rollback to the *bench* for the work done by a crashed player or a player which has left the cell. For this purpose *coordinator* must store message IDs sent by the `PTID`s that are not committed. When the commit message is sent by a player the corresponding storage is freed. So coordinator gives a rollback message to bench with `TTID`, `PTID`, `Start-MessId`, `End-MessId`. On receiving the rollback message bench removes the corresponding entries in order to maintain the consistency. So this recovery involves the processing at the coordinator and the bench.

6.3 Recovery at Bench

As the *coordinator* itself is a mobile host, but with some special properties, it may also crash and also leave the current cell. This situation is handled by *bench* by choosing another `MHID` as *coordinator* and removing the work done by the previous *coordinator*. This may degrade the performance. Therefore the *coordinator* is chosen from among the MHs which are expected to stay in the current cell for a duration longer than the expected completion time for a transaction.

7 Implementation Specifications

Figure 5 shows a sample display of *the bench* performing transactions. The specifications of attributes and poeration of the *bench*, the *coordinator* and the *mobile host* inlcuding that of the recovery actions are given below.

7.1 The bench

Attributes: List of `MHID`s, `TTID-CID` list, Action buffers per player per transaction.

Operations:

Create_MHIDs_with_Lifetime(){
    Randomly generate `MHID`s with lifetimes.
    Store them in a list.

```
ttid 86 given to 1345
ttid 49 given to 4652
ttid 4 given to 1125
ttid 44 given to 1997
ttid 86 stopped
ttid 86 is given to 4981
ttid 49 stopped
ttid 49 is given to 4877
ttid 4 stopped
ttid 4 is given to 3824
ttid 44 stopped
ttid 44 is given to 3877
ttid 86 commited
ttid 49 commited
ttid 4 commited
ttid 44 commited
time for ttid: 86 is 1147
time for ttid: 49 is 1023
time for ttid: 4 is 1130
time for ttid: 44 is 1063
```

Figure 5: Snapshot of Bench while performing transactions

```
}

Choose_Coordinator_and_Initiate_Transaction(){
    Choose a coordinator and assign a transaction to it.
    Store (TTID, CID) pair in a data structure.
}

Process_Received_Data(){
    If (message type == ROLLBACK)
        call rollback();
    If ((message type == COMMIT) and message from CID)
        call transcation commit();
    Else
        write to logtable();
}

Rollback(){
    If (partial rollback)
```

```
        Update the action buffer of the player by deleting the messages
        according to (RSMessID, RSLSN, REMessID, RELSN)
    If (total rollback)
        Delete the whole action buffer for that player
}

Transcation_Commit(){
    Commit the operations in the action buffers of the particular TTID
}

Write_to_Logtable(){
    Store the message in corresponding action buffer of TTID
}

Choose_Another_Coordinator(){
    If (no message from the coordinator till timeout)
        Choose another MHID as coordinator and assign the TTID to it.
        Update (TTID,CID) table.
}
```

7.2 The Coordinator

Attributes: List of Active MHIDs, List of Vacant MIDs, Recovery system, (TTID, PTID)-list, (PTID, CID)-list, PTID commit-list, CID, Lifetime, BenchID.

Operations:

```
Choose_Members_and_Start_operation(){
    Divide TTID to PTID and assign each PTID to a MHID.
    Update (TTID, PTID)-list.
}

Receive_Data_from_Bench(){
}

Receive_Data_from_Hosts(){
    If (message type==SPLIT_DELEGATE)
        choose another player()
        return
    If (message type==COMMIT)
        Update PTID commit-list
        Send data to bench()
}
```

```
Choosing_Another_Player(){
    Pick up a MHID from vacant MHs and assign PTID to it.
    Remove this MHID from the vacant MHs list.
    Get the status of old player from the bench.
    Inform the bench about this change.
    Send the status to this new player.
}

Send_Data_to_Bench (){
    If (TTID is finished)
        Send TTID commit to bench.
    Forward messages to bench.
}
```

7.3 The Mobile Host

Attributes: MHID, Lifetime, (PTID,CID)-list

Operations:

```
Local_Rollback(){
    Locally undo the action performed.
}

Rollback_to_Coordinator (){
    If (partial Rollback)
        Provide RSMessID,RSLSN,REMessID, RELSN to coordinator for rollback
    If (total rollback)
        Give PTID to rollback completely
}

Send_Data_to_Coordinator (){
    Process the operation()
    Perform the transaction and send the result to coordinator
}

Receive_Data_from_Recovery_System() {
    Receive the status from which it has to start transaction
}

Process_Operations() {
    Perform the assigned task to it.
}

Delegate(type){
```

```
        If (PTID is over)
            Send delegate message to CID
        Else If (leaving the cell)
            Send SPLIT_DELEGATE message to CID.
    }
```

8 Comparisons of the two models:

The kangaroo transaction model captures the mobility of a single node during the transaction processing. In Team Transaction model many hosts will group into a team for transaction processing. So the two transaction models are designed keeping in view of different situations. Kangaroo transaction model is silent about recovery scheme. If a crash occurs at any of the site then the work done is undone at all sites the host had traversed. Team model inherently supports recovery mechanisms, having partial rollback and total rollback if required. The time taken to commit a transaction in kangaroo model is understably less as compared to Team model since the message overheads in Team transaction model are high. But for long-lived transactions Team transaction model gives better results.

9 Conclusions

In this paper we presented two transaction models kangaroo transactions and team transactions. We presented the recovery mechanisms in both the models. The team transaction model has been proposed for the collaborative environments. It has been modeled for the applications which need a team effort to carry a task. Examples are collecting population data, product popularity survey, etc. The kangaroo transaction model captures movement of a single mobile transaction. This model is not meant for collaborative environments We have implemented the recovery schemes. Although transaction commit time in Kangaroo model is less compared to team transaction, the team transaction model considers the robust mobility

References

[1] Margaret H. Dunham, Abdelsalam Helal and Santosh Balakrishnan. A mobile transaction model that captures both the data and movement behavior, *ACM/BALIZER Journal on Special Topics on Mobile Networks and Applications (MONET)*, vol.2(2), pp.149-162, June 1997.

[2] M. M. Gore and R. K. Ghosh. Contention-free team transaction management and recovery on mobile networks with ad hoc groupings. In *the Proceedings of 4th ICIT 2001*, December 2001, pp 31-36.

[3] C. Pu, G.Kaiser and N. Hutchinson. Split-transactions for open-ended activities. In *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 26-37 VLDB Endowment, 1988.

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

[5] M.M. Gore and R.K. Ghosh. Recovery in distributed extended long-lived transaction models. In *Proceedings of the 6th International Conference Data Base Systems for Advanced Applications(DASFAA)*, pages 313-320, April 1999.