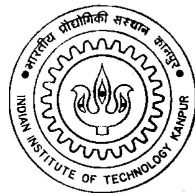
Implementation of a Collaborative Transaction Processing System on MANET

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

Shalini Varshney



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

April, 2002

Certificate

This is to certify that the work contained in the thesis entitled "*Implementation of a Collaborative Transaction Processing System on MANET*", by *Shalini Varshney*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

April, 2002

(Dr. R. K. Ghosh)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

The mobile computing environment poses its unique challenges to the existing transaction processing models, which fail to capture its features. The primary characteristics of such an environment are scarce network bandwidth, power resources and highly heterogeneous computing environment. Hence a transaction model has to adapt to frequent prolonged disconnections and accept the long-lived nature of the transactions to work successfully in such an environment.

In this work, we have developed and implemented a new transaction model called Team Transaction Model, that works through a collaborative effort of mobile nodes and does the processing of a transaction in a distributed fashion in an ad hoc wireless network. The model captures the abstract idea of a *team* as can be seen in any group effort in real life. It uses the team concept and defines a hierarchical structure using three classes of nodes - coordinator, player and data access agent. Player nodes work under the supervision of the coordinator. Data access agents have been introduced to decouple the transactional operations from the management of logging activities necessary for recovery purposes.

Acknowledgement

I take this opportunity to express my sincere gratitude to my supervisor Dr. R.K.Ghosh for his invaluable guidance. It would not have been possible for me to take this project to completion without his relentless support and encouragement. I consider myself extremely fortunate to have had a chance to work under his supervision. It has been a very enlightening and enjoyable experience to work under him.

I also wish to thank all the faculty members of the Department of Computer Science and Engineering for imparting their invaluable knowledge in course of my MTech program. I also extend my thanks to the technical staff of the department for maintaining an excellent working facility.

I would also like to thank my batch-mates who have made my stay in IIT Kanpur, the most memorable one. The name 'mtech2000' became a synonym to family where we all enjoyed working together and remained close in all thicks and thins. I find it hard to forget the "bulla sessions" I used to have with my friends in GH.

I would like to thank my parents and brothers for providing me necessary support and encouragement for building a good career and a bright future. Finally, I thank the Almighty to keep showering me with all his love and luck and grant me an opportunity to be here in one of the world's best educational institution.

Contents

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Related Work . . . 2
1.3 Organization of the Thesis . . . 4

2 Team Transaction **5**
2.1 Team Transaction with Bench . . . 5
2.2 Team Transaction without Bench . . . 6
2.3 Objectives of the Model . . . 6
2.4 Working of the Model . . . 7
2.5 Key terms . . . 9
2.6 Team Transaction using ACTA formalism . . . 9

3 Implementation **14**
3.1 System Architecture . . . 14
3.2 Communication Server . . . 18
3.3 Implementation of a mobile node . . . 18
3.3.1 Module Interfacing Manager(MIM) . . . 19
3.3.2 Communication Manager(CM) . . . 20
3.3.3 Transaction Manager(TM) . . . 21
3.3.4 Log Manager(LM) . . . 22
3.3.5 Recovery Manager(RM) . . . 23

4 Performance Study and Conclusions **25**
4.1 Case Study . . . 25
4.1.1 Time Measurement . . . 25
4.1.2 Variation with job size . . . 28
4.1.3 Variation with number of nodes . . . 28
4.2 Comparison with other transaction models . . . 29
4.3 Conclusion . . . 30

A Packet Types with Description **33**

B Implementation Specifications **35**
B.1 Data Access Agent . . . 35
B.2 Root/Coordinator . . . 36
B.3 Player . . . 38
B.4 Recovery Mechanism . . . 40

Bibliography **41**

List of Tables

4.1 Transaction time for a single node 27
4.2 Transaction time for a root with a player 27
4.3 Transaction time for a team of root, coordinator and player 28
4.4 Mobile Transaction Models I 31
4.5 Mobile Transaction Models II 32
A.1 Packet Type Description 34

List of Figures

2.1 A Cluster 7
2.2 Generalized Team Transaction 8

3.1 System Architecture 15
3.2 Thread Model of the System 19
3.3 In_Queue and Out_Queue 21

4.1 Job Size vs. Total Time 29
4.2 No. of nodes vs. Total Time 30

Chapter 1

Introduction

Computing while on move, using the smaller and portable computers is gaining wide acceptance among the professionals, emergency problem shooters, et al. Mobility is the buzz word of the day. It is possible to plug these portable computing devices with the wireless interface, and communicate amongst one another. In a mobile computing environment, a user can communicate, and access various shared services and shared resources remotely through the network at any point of time and from anywhere. But with the advent of new technologies, what comes in the package is fast growth of a new generation of software and considerable amount of pressure on the *compatibility* with the legacy technologies.

The infrastructure-less wireless or self-configurable networks are called Mobile Ad Hoc Networks. Such networks are useful in cases where no infrastructure is required or is not available and has capability of rapid deployment. Therefore ad hoc networks are considered to be extremely attractive for a number of application such as tactical communication facility in military, law enforcement, disaster and rescue operations. On commercial front, ad hoc networks can be used in personal area network, embedded computing applications, sensor dust, etc.

An ad hoc network is a spontaneous network formed by the groupings of a number of mobile devices. Each mobile host is assumed to have a fixed range within which it can broadcast messages. Any mobile device within the wireless transmission range

of another mobile host can listen to and capture the messages transmitted by later. Such an environment poses various limitations like disconnectivity, low bandwidth, high bandwidth variability, location dependent information, address migration, privacy, heterogeneous network, security risks, and low battery power.

1.1 Motivation

When mobile users issue data queries, the host transmits a message and thus performs a transaction. Therefore, a suitable transaction processing model is an important challenge for data management in mobile computing environment. The mobile computing involves some unique problems that need to be addressed by the transaction model proposed. The most important issues which are typical to mobile computing include the devices frequently switching between the *"off-the-net"* and *"on-the-net"* status, due to the provision of disconnectivity - voluntarily or involuntarily. Transactions become long-lived due to the frequent as well as prolonged disconnection periods. All conventional transaction models assume underlying network to be robust though nodes may crash occasionally. Therefore, the need to formulate a new transaction model suitable specially to mobile computing environment assume special significance.

A new transaction model called Team Transaction Model has been proposed in [4] for infrastructured mobile wireless network with ad hoc grouping. This model exploits the features of the ad hoc environment to form a collaborative effort by all the nodes in the vicinity to form a team and serve the query request launched by one of the mobile nodes. The model focuses on the issue of *reliability* rather than *availability*. The present investigation examines the issues and related problems in extending and implementing the team transaction model on a purely ad hoc network.

1.2 Related Work

A mobile transaction is structured as a distributed transaction. In presence of an infrastructured mobile wireless network, the transaction is partly executed on

mobile nodes and partly on fixed nodes. Usually, a mobile node performs the task of initiation, while actual transaction is performed by the static nodes, because database is typically located at the nodes.

However, the mobile infrastructure produces significant additional challenges to transaction processing. The wireless networks allows a limited bandwidth, thus making network bandwidth a scarce resource. Data transmission and local computation drain the battery power of a mobile unit. Hence power consumption becomes serious constraint for transaction processing. Frequent disconnections due to a voluntary step to save power, or movement of the node out of the network's range, or running out of power pose several other challenges. Finally, mobile nodes which vary widely in their capabilities, provide a highly heterogeneous computing environment. Transaction processing in such a heterogeneous environment is another a big challenge.

A few models have been proposed to capture the mobile specific features. Dunham [3] suggests a model called Kangaroo Transactions Model, where mobile units are the basic unit of computation that initiate a transaction. This is further extended to the data source by the data access agent. The data access agents reside on mobile support station *(MSS)* and work on behalf of the mobile units which lie in the range of the host *MSS*. The transaction logically hops from one *MSS* to another with the movement of the mobile unit. However the model does not talk about recovery.

Chrysanthis [1] considers mobile transactions a special type of multidatabase transaction, and introduces the additional notions of reporting and co-transactions. [7] introduces transaction proxy concept. A proxy runs at *MSS* corresponding to each transaction and ensures periodic backup of the computations done at the mobile hosts. Pitoura and Bhargava [8] propose a model wherein they consider mobile transactions an issue of maintaining consistency in a global multidatabase which is divided into clusters. In [9] the model emerges from semantics-based transaction concepts, and views mobile transactions as a problem of managing cache coherency

and concurrency in a distributed database. Yeo and Zaslavksy's model [11] suggests that mobile transactions should support the notions of long-lived transactions and sagas, and that mobile transactions are a special type of multidatabase global transaction. Pro-motion [10] provides for disconnected transaction processing, and allows the local management of transactions via constructs called compacts.

Although so many new models based on varying concepts have been proposed, no single best approach has emerged and the issue is still being explored by the researching community. In this work, we propose a model that could address to the issue of transaction processing along with recovery technique in a mobile wireless network, keeping the model as simple as possible, at the same time allowing extended long-lived transactions.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows:

Chapter 2 talks about the Team Transactions - with bench and without bench. It gives the objectives of the model, and provides hints about the terminology used. Later, it also reviews mathematical axioms which forms the basis of the team transaction.

Chapter 3 gives an overview of the System Architecture of the mobile transaction processing model and then gives implementation details. Transaction processing as well as recovery techniques have been covered in details.

Chapter 4 gives case studies done with the system developed herewith and talks about performance. It concludes with a summary of the work and comparison of the team transaction model with the existing models.

Chapter 2

Team Transaction

A team transaction is composed of several nodes working in a collaborative fashion with a common objective of completing a query request issued by one of the nodes. It has a three tier structure just like teams of any sport, namely, the bench, the coordinator on the field and the individual team members - players. The basic idea has been discussed in [5]. In this chapter first we examine a straight forward extension of the idea and then present the theoretical frame.

2.1 Team Transaction with Bench

Initially, the Team Transaction model was proposed for an infrastructured wireless network environment. The *bench* forms the top level of control, in transaction management. It lies on the mobile support stations and has the support of the fixed network. This is where the database is housed, may be in a distributed manner. The bench because of its inherent reliability, is used to store stable logs of transactional activities of team(s), and to provide recovery support.

The next one in the hierarchy is the *coordinator* mobile host that initiates the transactional activities, distributes jobs to team members and finally terminates the transactional activities. It accesses the bench to log its actions and the messages received from the team members. It reaches other team members through a diameter 2 ad hoc network. The lowest level member in the hierarchy is *player*. These are

the mobile hosts that volunteer to a coordinator for processing a transaction and delegate the responsibility of the work done to the coordinator. Thus a team of mobile nodes make a collaborative effort to process a transaction.

2.2 Team Transaction without Bench

Extending Team Transaction Model to a completely ad hoc network requires a couple of modifications to suit the environment. Since the environment is purely ad hoc, there is no participation of fixed network. Thus, such a version of team transaction can be alternatively referred to as ***"team transaction without bench"***. In the original model of team transaction, the bench provides recovery support. Since the bench is eliminated in the new model of team transaction, another set of nodes called Data Access Agent has been introduced to ensure support of recovery guarantees. DAAs are supported to be less mobile and more robust among the set of nodes forming the ad hoc network whereas the other two members of the hierarchy remain same.

The new model allows multiple levels of hierarchy in the formation of team, hence a generalized model. It allows division of work at many levels, forming a tree like structure. With the introduction of this generalized structure and in the absence of fixed network to act as backbone, the task of transaction processing and recovery becomes more challenging. In the rest of the thesis we focus our discussion only on this team transaction model without bench.

2.3 Objectives of the Model

The very idea of formulating a new model for transaction processing emerges out of the new problems posed by the mobile ad hoc networks. The new model tries to answer some of them which are stated as below:

- Transaction should be long-lived.
- Transaction should support distributed and collaborative computing.

- Transaction should be able to survive through any number of crashes of the nodes forming a part of the team.

2.4 Working of the Model

Three primary entities of the *Team Transaction* model are:

1. **Coordinator**: A transaction that supervises and coordinates its work amongst a few other nodes that together forms one team.

2. **Player**: A sub-transaction that performs a piece of work assigned to it by the coordinator.

3. **Data Access Agent**: It is a supposedly more robust node with low mobility that provides data access to the coordinator. It is also responsible for logging information required to recover a crashed coordinator. The node is also supposed to be alert and determine immediately whenever the coordinator crashes and delegates incomplete work to another node. Therefore, the DAA always keeps track of the coordinator.

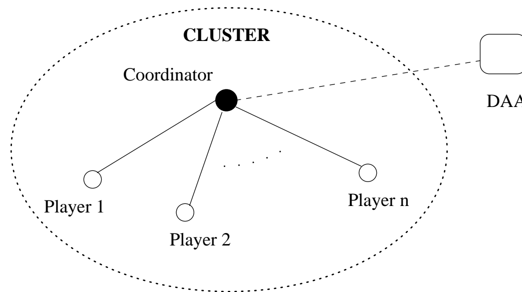


Figure 2.1: A Cluster

The team transaction is initiated by a node that issues a database query which is big enough for the node to execute single handedly, as well as distributive in nature.

A team transaction springs up to handle such a query. The node looks for players that are eager to volunteer for completing the job. Each of the player houses a sub-transaction that reports to the coordinator on completion, the coordinator being the first. As soon as the coordinator is ready to spawn players, it also looks around for a DAA, a node required for logging the necessary info that plays a vital role in recovery, in case of crashes.

A player node is required to periodically report the updates done by it to its coordinator. The player can not commit the changes to the database, but only delegate the commits to the coordinator. It is up to the coordinator whether it finally commits the changes to the database. Thus the right of committing updates to the database rests only with the coordinator.

This model can further be extended. If a player finds the work assigned to it to be too big, it can break down the transaction into independent blocks of work and then starts a team transaction at lower level. The upper level coordinators may not at all be aware of this. The generalized model is depicted in Fig. 2.2

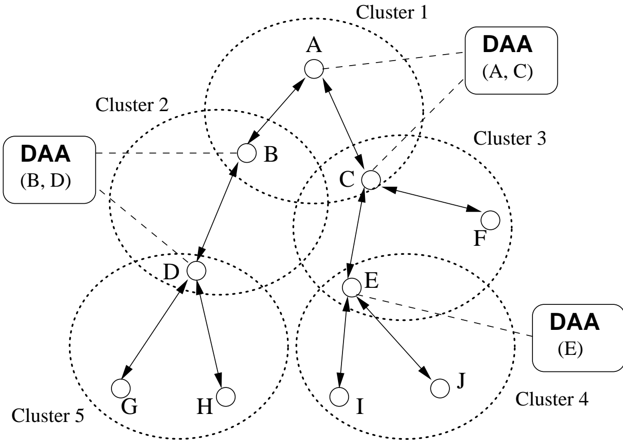


Figure 2.2: Generalized Team Transaction

2.5 Key terms

A brief explanation of the various terms used in the model follows:

- **Cluster, Coordinator and Player Transaction**: A team transaction consists of one or more subsets of transactions called *clusters*, where each cluster contains exactly one *coordinator* transaction and one or more *player* transactions. Though it is possible that a single transaction may be a player transaction to one cluster while being a coordinator transaction to lower level cluster.

- **Significant Events**: These are the transaction management events that may occur more than once during execution of the transaction. Various such events are *begin*, *commit*, *abort*, *spawn*, *work*, *accept*, *split* and *kill*.

- **Work Event and Work Set**: *Work event* is the event of the coordinator transaction assigning work to its player transactions, and the set of operations that is being assigned constitutes the *work set*.

- **Report Event and Report Set**: *Report event* is the event of the player transaction delegating the work done by it to its coordinator transaction. The set of operations done by the player constitutes the *report set*.

- **Accept Event and Accepted Report Set**: *Accept event* is the event of the coordinator transaction accepting the player transactions work after the *report event*. The coordinator may accept a part or whole of the reported set or may reject it. The accepted set constitutes the *accepted report set*.

- **Split Event**: This event occurs when a node delegates its unfinished work to some new node and walks out of the team. A player as well as a coordinator can split at any point of time.

2.6 Team Transaction using ACTA formalism

In this section, we present a formalization of the transaction model using a set of axioms that comply with the ACTA [2] framework of transactions, along with brief

explanations about these axioms. For convenience, we introduce a few notations:

t, t' $\rightarrow$ any transaction (player or coordinator),

t_0 $\rightarrow$ the root coordinator transaction,

t_p, $t_{p'}$ $\rightarrow$ any player transaction other than the root

Coordinator(t) $\rightarrow$ the cluster coordinator of the player transaction t

Player(t) $\rightarrow$ the set of all cluster players of a coordinator transaction t

α, β are any operations, and

DB $\rightarrow$ set of database objects.

AXIOMS:

1. SE_{t_0} = {Begin, Commit, Abort, Spawn, Work, Accept, Split, Kill}
2. IE_{t_0} = {Begin}
3. TE_{t_0} = {Commit, Abort}
4. SE_{t_p} = {Spawn, Commit, Abort, Report, ReportAbort, Work, Accept, Split, Kill}
5. IE_{t_p} = {Spawn}
6. TE_{t_p} = {Commit, Abort, Kill}
7. All transactions satisfy the four fundamental axioms about the initiation and the terminal events[2].
8. $\text{ViewSet}_{t_p} = \bigcup_{Player(t_p)}$ ((Last AcceptedReportSet) $\bigcup$ Last WorkSet from Coordinator(t_p) $\bigcup$ (Previous ViewSet))
   Initially, $\text{ViewSet}_{t_p} = \phi$.
9. $\text{ViewSet}_{t_0} = \bigcup_{Player(t_0)}$ ((Last AcceptedReportSet) $\bigcup$ (Previous ViewSet) $\bigcup$ DB)
   Initially, ViewSet_{t_0} = DB
10. $\text{ConflictSet}_t = \forall t' \; \forall \beta \; \forall ob \; ((t' \neq t) \; \bigwedge \; \beta_{t'}[ob] \in \text{ViewSet}_{t'} \; \bigwedge \; \text{InProgress}(\beta_{t'}[ob]))$

11. $\forall ob\ \exists t\ \exists\alpha\ \alpha_t[ob] \in H \Rightarrow$ (ob is atomic)

12. $\text{Commit}_t \in H \Rightarrow \neg(tC^*t)$

13. $\exists ob\ \exists\alpha\ \exists t\ \text{Commit}_{t_0}[\alpha_t[ob]] \in H \Rightarrow \text{Commit}_{t_0} \in H$

14. $\text{Commit}_{t_0} \in H \Rightarrow \forall ob\ \forall\alpha\ \forall t\ (\alpha_t[ob] \in \text{AccessSet}_{t_0} \Rightarrow \text{Commit}_{t_0}[\alpha_t[ob]] \in H)$

15. $\text{Commit}_t \in H \Rightarrow (\forall t_p\ (\text{Coordinator}(t_p) = t \Rightarrow (\text{Commit}_{t_p} \bigvee \text{Kill}_t[t_p]))$

16. $\text{Commit}_{t_p} \in H \Rightarrow (\forall ob\ \forall\beta\ \forall t\ (\beta_t[ob] \in \text{ViewSet}_{t_p} \Rightarrow \text{Report}_{t_p}[\ \text{Coordinator}(t_p), \beta_t[ob]]$ $\in H))$

17. $\text{Abort}_{t_0} \in H \Rightarrow\ \forall ob\ \forall\alpha\ \forall t\ (\alpha_t[ob] \in \text{AccessSet}_{t_0} \Rightarrow \text{Abort}_{t_0}[\alpha_t[ob]] \in H)$

18. $\exists ob\ \exists\alpha\ \exists t\ \ \text{Abort}_{t_0}[\alpha_t[ob]] \in H \Rightarrow \text{Abort}_{t_0} \in H$

19. $\text{Abort}_{t_p} \in H \Rightarrow\ \forall ob\ \forall\beta\ \forall t\ \beta_t[ob] \in ViewSet_{t_p} \Rightarrow \text{ReportAbort}_{t_p}[\ \text{Coordinator}(t_p)]$

20. $\text{ReportAbort}_{t_p}\ [\text{Coordinator}(t_p)] \Rightarrow\ \forall ob\ \forall\beta\ \forall t\ (\beta_t[ob] \in H \Rightarrow \text{Report}_{t_p}[$ $\text{Coordinator}(t_p), \beta_t[ob]]\ \bigwedge\ \neg\ \text{Accept}_{Coordinator_{(t_p)}}\ (\text{Report}_{t_p}[\ \text{Coordinator}(t_p), \beta_t[ob]])$

21. $\text{Spawn}_{t_{Coordinator(t_p)}}[t_p] \in H \Rightarrow (t_p\ \mathcal{WD}\ t_{Coordinator(t_p)})\ \bigwedge\ (t_{Coordinator(t_p)}\ \mathcal{CD}\ t_p)$

22. $\text{Kill}_t[t_p] \in H \Rightarrow t$ is coordinator transaction of some cluster $\bigwedge$ t_p is a player transaction of that cluster.

23. $\forall t_p(\ \text{Coordinator}\ (t_p) = t)\ \bigwedge\ (\ \text{Accept}_t[\ \text{AcceptedReportSet}_{t_p}]) \Rightarrow (\ \text{AcceptedReportSet}_{t_p}$ $\subset \text{ReportSet}_{t_p}\ \bigwedge\ (\ \forall\alpha\ \forall t'\ \forall ob\ (\alpha_{t'}[ob] \in \text{AcceptedReportSet}_{t_p} \Rightarrow \text{Responsible}[t, \alpha_{t'}[ob]])))$

24. $\forall t_p\ \exists ob\ \exists\alpha\ (\alpha_t[ob] \in \text{AcceptedReportSet}_{t_p} \Rightarrow (\forall\beta\ \ \beta_t[ob] \in \text{ReportSet}_{t_p} \Rightarrow$ $\beta_t[ob] \in \text{AcceptedReportSet}_{t_p}))$

25. $\text{Report}_{t_p}[t_{Coordinator(t_p)}, \text{ReportSet}_{t_p}] \Rightarrow (\ \text{ViewSet}_{t_p} = (\text{Previous ViewSet})_{t_p} -$ $\text{ReportSet}_{t_p})$

26. $\forall ob\ \exists\beta\ \forall\alpha\ \forall t_p\ ((\beta_{t_p}[ob] \in H\ \bigwedge\ \beta_{t_p}[ob] \rightarrow \alpha_{t_{Coordinator(t_p)}}[ob]) \Rightarrow (\ \text{Report}_{t_p}[$ $\text{Coordinator}(t_p),\ \text{ReportSet}_{t_p}] \in H\ \bigwedge\ \beta_{t_p}[ob] \in \text{ReportSet}_{t_p}\ \bigwedge\ \text{Report}_{t_p}[$ $\text{Coordinator}(t_p),\ \text{ReportSet}_{t_p}] \rightarrow \alpha_{t_{Coordinator(t_p)}}[ob])$

Axioms 1 to 6 specify the significant events, initiation events and termination events of the coordinator and player transactions. A player uses `ReportAbort` to report an abort to its coordinator. `Spawn` when invoked by a player, spawns a player for it. Axiom 7 refers to the axioms mentioned in ACTA formalism.

Axiom 8 defines the `ViewSet` of any non-root transaction to be the ordered union of (i) the report sets from any of the players, (ii) `WorkSet` handed to it by its coordinator and (iii) the previous `ViewSet`. As axiom 9 specifies the `ViewSet` of a root transaction is similar to that of the players accept its initial `ViewSet` is DB. Axiom 10 specifies the `ConflictSet` of a transaction to be all `InProgress` operations.

Axioms 13 and 14 talk about the failure atomicity of the root coordinator transaction. Axiom 15 states that a transaction can commit only when all its player transaction commit or one or more of them are explicitly killed by it. Axiom 16 talks about the commit of a player which implies it has reported all its operations in the `AccessSet` to its coordinator. Axioms 17 and 18 specify the failure atomicity of a coordinator transaction regarding an abort event. If a coordinator transaction aborts then all its operations also abort and vice versa.

Axioms 19 and 20 state that when a player transaction aborts, it informs its coordinator through the event `ReportAbort` and none of the operations are accepted by the coordinator. Axiom 21 states that if the coordinator aborts, all the player transactions that have not yet committed, should also abort. Thus player transactions are weak abort dependent upon its coordinator.

Axiom 22 talks about the `Kill` event that can be issued by a coordinator for one of the players in the same cluster. Axioms 23 states that coordinator can accept a subset of the `ReportSet` of a player. Axiom 24 specify that the if some operation by a player on an object is accepted, than **all** operations on the object by the player **must** be accepted. Axiom 25 states that once a player reports its operations to the coordinator, they are removed from the `ViewSet` of the player. Axiom 26 defines a

restriction on the coordinator regarding the `WorkSet` which once handed over to a player cannot be accessed by the coordinator till the player reports back.

Chapter 3

Implementation

In this chapter, we discuss about the implementation of the generalized team transaction model without bench. The study of model in a formal setting in chapter 2 helped us in formulating the overall design of system architecture, and subsequent implementation of the basic entities, namely the coordinator, the player and the data access agent. The implementation of various components of each entity and the recovery algorithms have been discussed in details after presenting the system architecture.

3.1 System Architecture

In order to capture all the features of the *Team Transaction Model*, and interfacing of its basic entities, a suitable system architecture has to be worked out. The architecture proposed for the underlying system is described in this section. The system has to be generic enough to reflect the necessary features of all the three primary entities, namely, the *Coordinator*, the *Player* and the *Data Access Agent*. The main components of the system proposed are described below, one by one.

As shown in Fig. 3.1, a node meets the system requirements through a number of component modules. Depending upon the specific role that a node has to perform, the system support requirements at any node may be of two different types. A node which acts as a DAA is responsible for recovery of crashed nodes. Any other node

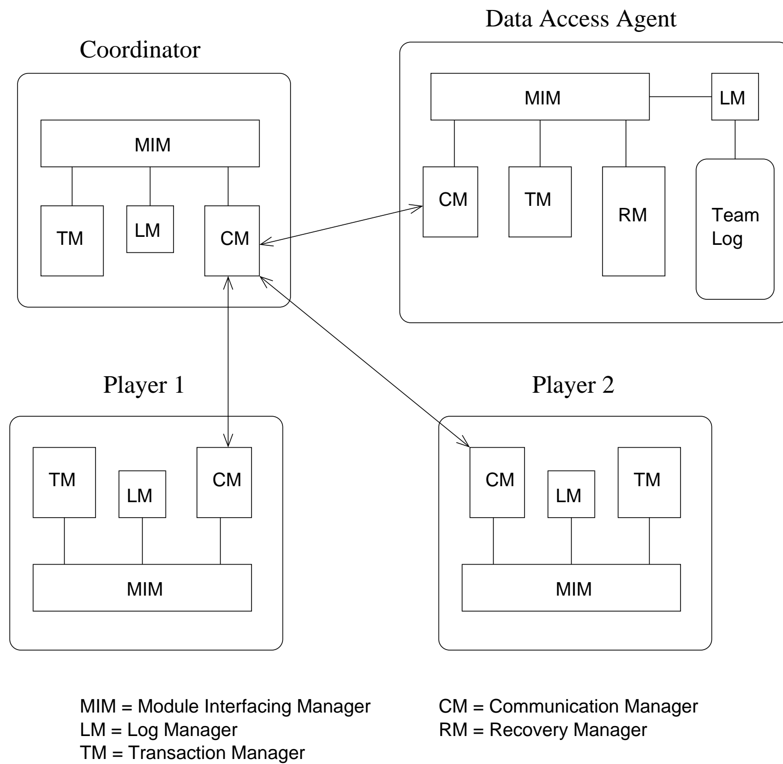


Figure 3.1: System Architecture

is termed as simple mobile node. A simple mobile node can issue queries, start new transactions, or help other mobile nodes in processing small parts of a transaction as a player node.

A brief description of the all the constituent entities of a node follows:

- ***Module Interfacing Manager (MIM):***

  The Module Interfacing Manager provides a common interface for interactions among all other components of the node. The MIM is responsible to keep track of all the transactions running on the node along with their roles in the transactions. It may happen that a coordinator sub-transaction of one

transaction is concurrently running on the node with a player sub-transaction of another transaction on the same node.

The job of DAA is exclusively for logging the operations of coordinators. Hence MIM does not have the responsibilities concerning execution of transactions. However the major responsibilities of providing recovery guarantees in the event of a crash is borne by a DAA. Therefore, MIM at a DAA has to keep track of all the coordinator for whom the DAA is serving, and has to immediately detect crash of a coordinator and find a new replacement, by picking one from the set of willing nodes.

- ***Communication Manager (CM):***

The Communication Manager is basically responsible to facilitate communication and data transfer across nodes. The CM of a mobile node deals with packets arriving at its end in FIFO manner. It reads from the net the incoming packets and passes them to the application manager for further operations. The CM is responsible for pushing the outgoing packets on the net.

- ***Transaction Manager (TM):***

The primitives required by a (sub)transaction to execute the given set of operations are provided by the transaction manager. It provides the basic primitives related to transaction, report and spawn to execute the application. It ensures the logging of all operations including the significant events, by staying in close association with the log manager and communication manager.

The TM issues various primitives depending on situation present in the surrounding environment. It creates work sets on the basis of number of nodes available in the vicinity, or does the whole work single handedly in the absence of players. The TM of a coordinator has the power of discretion of either rejecting or accepting (completely or partially), the work reported

by its player transaction. Killing a player transaction, splitting its work and delegating to some other node, calling an abort to its own work or committing are also at the discretion of the TM which is done on the basis of the existing state of the database, or location parameters of the node.

- ***Log Manager (LM):***

  The operations being executed by a player transaction on various database items are logged by the Log Manager locally at each node. In addition to this, the coordinator, has to maintain the log of its team. The log of a team includes the work done by itself as well as by all its players lying in the same cluster. The players are required to flush their logs intermittently to their coordinator to keep the coordinator updated of its latest work. Besides that, the team logs are also flushed to the DAA of the team, i.e. the cluster. This team log is maintained by the DAA on its stable storage. It is this log that is utilized by the DAA in recovering a crashed coordinator.

  The LM is accessible through the interface provided by the MIM. TM works in close association with the LM as each and every action performed by the TM has to be logged.

- ***Recovery Manager (RM):***

  The Recovery Manager exists at the special nodes called *Data Access Agent* only. Since the DAA is the node that possesses the team logs of given coordinator(s), it is the DAA's responsibility to recover the crashed coordinator.

  The recovery algorithm followed is based on the ARIES method proposed by Mohan et al [6]. As soon as the crash of a coordinator is detected, search for a new coordinator begins. Meanwhile, the RM picks the log from stable storage. The log gives information about the database and the job that was in progress at the time of crash at the concerned coordinator. Using this info,

the RM analyses the log, regains a consistent state of the database, a state nearer to the time when the node crashed and assigns the rest of the work to the newly elected coordinator. This coordinator then carries the work to the stage of completion.

3.2 Communication Server

In the next two sections, we would discuss the implementation of the system as designed on the guidelines discussed above. In our implementation we have used the *Communication Server* to simulate the ad hoc environment thus facilitating the routing of packets from one node to the other, be it a unicast, multicast or broadcast.

It keeps track of all the live mobile nodes, by maintaining a table of all the live nodes. An entry is picked up from the table whenever it is the destination of a packet arriving at the server. A new entry is added in the table when a node comes up and registers with the server. The server either directs the packets to a particular node or broadcast it to all, as per the destination mentioned in the packets.

3.3 Implementation of a mobile node

A mobile node has been implemented using a pluggable architecture, where *communication manager(CM)*, *recovery manager(RM)*, *transaction manager(TM)* and *log manager(LM)* can be plugged into the *module interfacing manager(MIM)*, depending on whether the node is a simple mobile node or a DAA. Each node is a separate process and these communicate through sockets as the underlying IPC mechanism. The reference to each of the entities is kept with the MIM, and provide an interface through which all entities interact amongst themselves.

A new node is defined either as a simple node or as a DAA, right at the time of creation. When a node comes up, its module interfacing manager gets initiated which initializes all other vital entities of the node. The MIM brings up the main thread accordingly named as *job allocator thread*. A node spawns a new thread for

each new work of a (sub)transaction appearing on the node. These threads have been named as *worker threads.* Fig. 3.2 gives the thread model of the implementation. The implementation details of each of the entities follows in the rest of the section.

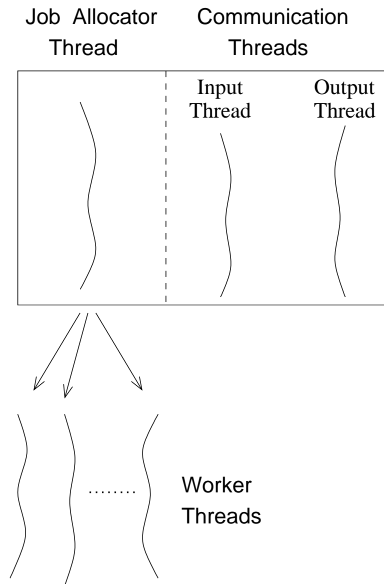


Figure 3.2: Thread Model of the System

3.3.1 Module Interfacing Manager(MIM)

As a node comes up, first the module interfacing manager gets created and then it creates all other required entities. In case the node is DAA, the RM is also created. It also brings up a front end for the node, where one can create a new application and launch a query as well as view messages, incoming and outgoing, both. The front end for a simple mobile node is designed different from the one of a DAA to visually indicate the difference.

It also creates two queues namely, *input queue* and *output queue* that it uses in conjunction with the CM for storing packets (Fig. 3.3.2). We would talk more about these queues in the subsection of CM.

The module interfacing manager starts a main thread called *job allocator thread*. Its main functions can be stated as:

- Start a new application whenever the user enters a query through the front end and assign it a unique team transaction id *(TTID)*.
- Process all the incoming packets and direct them to the concerned entity for further processing.

However, in case of a DAA, the work of the main thread differs to some extent, in the sense that DAA cannot start a team transaction nor can it participate as a player, etc. in a team transaction. Hence in case of DAA, the main thread has been named as *DAA thread*.

Primarily, the MIM gives an interface to all other entities of the node through which these entities can talk to each other and pass the semi processed data. It also keeps track of all the applications and the TTIDs running on the node.

3.3.2 Communication Manager(CM)

The Communication Manager is mainly responsible for enabling the node to converse with the outside world. It does so by sharing two queues named as *input queue* and *output queue* with the MIM.

The CM spawns two threads - *input thread* and *output thread* (Fig. 3.2) whose work is to constantly monitor the *input queue* and *output queue* respectively. Whenever a packet is to be sent out, it is placed by the MIM in the *output queue* which is later put on the network by the *output thread*. Similarly, whenever a packet arrives at the node, the *input thread* queues it in the *input queue*. It is then read by MIM. The packets are treated in FCFS manner (Fig. 3.3.2).

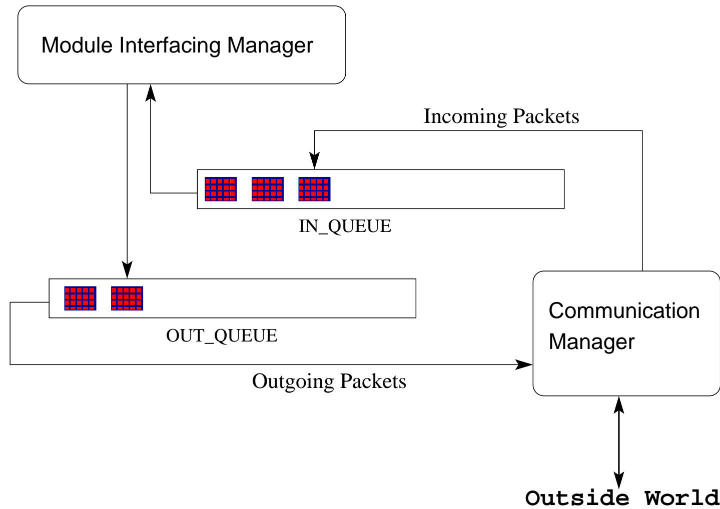


Figure 3.3: In_Queue and Out_Queue

3.3.3 Transaction Manager(TM)

The role of Transaction Manager is the most important one. It is here only that the work assigned to the node through a (sub)transaction is executed. The work is distributed, if it can be done in a distributive fashion and if nodes willing to volunteer as players are available in the vicinity. In the absence of any of the above mentioned conditions, work is done by the node itself.

After getting players and a DAA, the node changes its status to a coordinator and assigns work to each of the players. The node also creates entries in a *Pending Job Table* for each player where it maintains all the relevant data of each player transaction. A player is identified by a unique player transaction id *(PTID)*, assigned to it by its coordinator. Whenever, a player reports a *commit*, *abort* or *rollback*, the table is modified to reflect the status. In this process, a node keeps flushing its log to DAA (if coordinator or root) and to its parent node, after regular intervals.

To execute a work block locally, the TM spawns a thread named *worker thread* which is also assigned a unique PTID. Generally, it does so in absence of players.

Once a player completes its work and reports back to the coordinator, its up to the coordinator to accept or reject the work. If any of the players fail to complete its job, its coordinator assigns the same job to other player or does it on its own.

The TM maintains various timers which aid in its functioning. These timers are:

- *DAA Timer*: To wait for a response from DAA, failing which it re-broadcasts its request for DAA.
- *Response Timer*: To wait for response from players, failing which node works on its own.
- *Log Flush Timer*: To periodically flush its log to its coordinator.
- *Player Response Timer*: To detect if player has crashed.
- *New Player Timer*: Used when a piece of work has to be re-assigned as the earlier player transaction failed.

3.3.4 Log Manager(LM)

The Log Manager is primarily responsible for maintaining all the logs of various (sub)transactions running on the node. In addition, if the node plays role of a coordinator in a team transaction, it has to maintain logs of all the player nodes, and append them to its own log. This log is also then periodically flushed to the DAA of the node. However, a player is not assigned a DAA, because a player is given a small block of work that does not require to be logged by a DAA, and it is not worth an effort of recovery, if the player happens to crash.

In case of a DAA, the logs are stored on stable storage, as these logs are used for recovery of a crashed coordinator or root of a team transaction. While, logs are identified at a simple node through the TTID, in case of DAA, the logs are identified by a combination of TTID and the system id of the node, as the same DAA may serve more than one coordinators of a single team transaction.

The LM logs two types of log records. These are listed below with brief description:

- *UPDATE*: gives update info of a database item, by providing name of the database item, value before update and value after update, and new state id.
- *SIGEV*: depicts significant events that occurred during transaction processing, on the particular node.

The log of a root and a coordinator also contains necessary info that hints about the work being done by the node - the database items, it is operating on, and the instructions being assigned to it as job. This info is required for recovery. In our simulation, we have assumed the Network File System as the global database from where, database items can be read. So the root logs the database filename while a coordinator logs an image of all the database items it was given by its parent.

3.3.5 Recovery Manager(RM)

The mobile nodes being more susceptible to crashes, recovery becomes particularly more essential. The recovery algorithm proposed here is developed on the framework provided by one of the well known recovery algorithms - **ARIES**. The RM only exists on a DAA, as recovery can only be done by a DAA. It keeps track of all the coordinators it is serving, and detects immediately, as soon as a node crashes. In case of a node getting crashed, it first broadcasts a packet informing all of the node crashed. The log of the node, up to the last *Log Flush* received from the node, is read from the stable storage and processed as mentioned below:

1. The log is first put to an **Analysis Phase** where the log is scanned and a *Transaction Table* is created that gives info about all the player transactions of the coordinator in question and their status as reflected in the log.
2. The next one is the **Redo Phase**. Using the *Transaction Table* created in first phase, the work done by sub-transactions is either rejected in case a sub-transaction failed to commit, due to an abort or kill etc., or was still active at the time of crash. Else, if the player transaction committed, its updations are committed on the database items. The log is scanned backwards thus updating

the database items by only the last modified value. Hence, we optimize on the total number of updates.

3. Meanwhile a new coordinator is searched for, to take the half done (sub)transaction to completion.

4. The new coordinator is then assigned the modified database items, and the still-to-be-done operations set. It is then asked to proceed forward. The new node processes the (sub)transaction ahead in the similar fashion, as is done in normal execution.

We have used a modified algorithm for the recovery purposes, so as to be able to retain the latest consistent stage of the database just before the crash of one of the team members of a transaction. Thus we wish to loose the least possible work of a mobile node keeping in view the long-lived nature of transaction.

Chapter 4

Performance Study and Conclusions

4.1 Case Study

In this chapter, we describe the experiments conducted to measure performance issues in the *team transaction model*. The emphasis of our study is in investigating long-lived nature of transaction and the transaction which consists of large number of operations. The simulation environment we used did not have adequate physical components or software support for investigating many concurrently running transactions. Therefore, it was not possible for us to study the evaluation of performance for many transactions. The certain time for processing of a typical transaction in mobile ad hoc environment, in three primary cases, namely

- Single Node
- A root and a player
- A root, a coordinator and a player

Our study attempts to focus on the trend of the time taken by a typical transaction to complete depending on factors.

4.1.1 Time Measurement

Before displaying the measurements, we lay down the framework on which the experiments were conducted.

- ***System Specification:***

  All experiments have been performed on PCs with a 342 MHz Pentium II processor and 95 MB RAM. We have simulated one mobile device on one PC.

- ***Job Specification:***

  The job that was used as an example for transaction processing, consists of 600 instructions, where each an operation is a simple arithmetic operation such as an addition, a subtraction, a multiplication or a division.

- ***Time Breakup:***

  In the experiments, we measure the time taken for the completion of a typical transaction. The total transaction comprises of:

  - *Computation Time:* We define computation time as the time consumed by a mobile node in doing computation of the arithmetic operations.
  - *Distribution Time:* Distribution time is the time taken by a coordinator or root in distributing the job to a player.
  - *Communication Delays and Idle Time:* Communication delays involve time consumed by mobile nodes while communicating to each other. Idle time at a node is the time that a node sits idle doing nothing, as it is waiting for its player(s) to commit.

Typical Scenarios

We have taken the measurements for three different scenes, that pictures a mobile node in all the three roles: root, coordinator and player. The typical scenarios along with the corresponding time measurements are mentioned as follows:

1. **Single Node**

   This is the case where there is no other node in the vicinity of a mobile node where a transaction gets initiated. That is, a root node completes the transaction single handedly.

Total Time	Computation Time	Communication Time (DAA)
2829ms	1445ms	218ms

Table 4.1: Transaction time for a single node

2. **Root with a Player**

   This is the case the root node finds a single mobile node in its vicinity to share the load of a transaction that got initiated at the root node. The non-root node acts as a player to the root, forming a two-member team.

Node	Total Time	Computation Time	Distribution Time	Communication Time (DAA + Player)
Root	7896ms	812ms	500ms	493ms (283 + 210)
Player	-	866ms	-	-

Table 4.2: Transaction time for a root with a player

3. **Root, Coordinator and Player**

   This is the third case where we can find a generalized model of the team transaction. The root node finds a player to form a team in its cluster, and the player in turn enlists the support of another node in its cluster and becomes the coordinator to that player.

Node	Total Time	Computation Time	Distribution Time	Communication Time (DAA + Player)
Root	7187ms	662ms	463ms	364ms (206 + 158)
Coordinator	-	205ms	197ms	239ms (186 + 53)
Player	-	408ms	-	-

Table 4.3: Transaction time for a team of root, coordinator and player

Looking at the performance statistics in the three cases, we find that the total computation time almost remains the same, though it gets distributed amongst the root, coordinator and players, and is done parallely. The total time taken is the time in which transaction gets processed. The communication time includes the time that a node took to communicate to find its players and DAA, if any.

4.1.2 Variation with job size

In this experiment, we have tried to observe the trend that exists between the total time taken to process a transaction and the actual size of the job in terms of the total number of instructions, assigned to the root. The experimental setup has three nodes, acting as root, coordinator and player, respectively.

The trend in the Fig. 4.1 shows that when the job size is small, due to the distribution time, communication delay, etc. the total time taken is large. But as the job size increases, the computation time increases and the contribution of other overheads reduces in the total time taken. The trend that we found is as per the expectations. Thus, we can say that larger the size of the job, better would be the performance of the model.

4.1.3 Variation with number of nodes

We conducted this experiment to study the trend that exists between the total time taken in processing a transaction and the number of mobile nodes involved

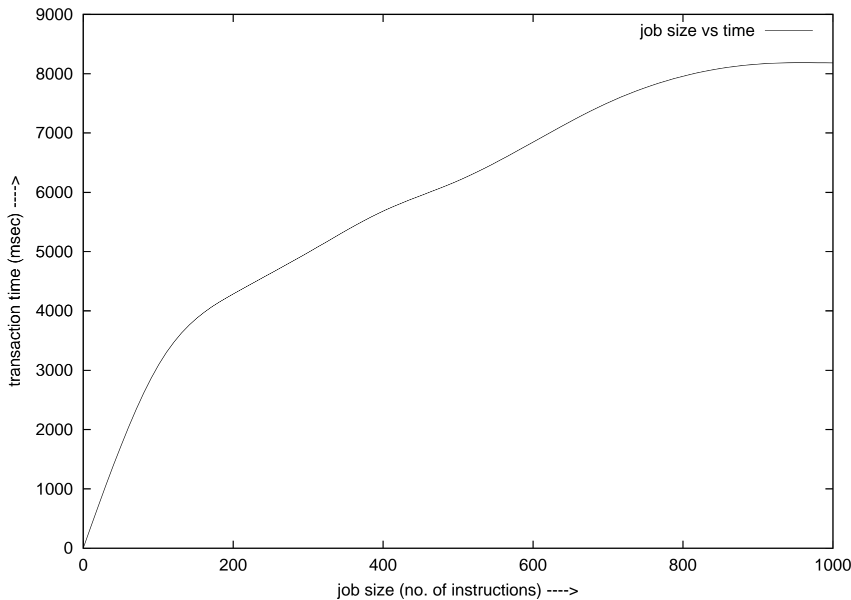


Figure 4.1: Job Size vs. Total Time

in the transaction. The sample job that was given to the nodes consisted of 600 instructions.

The trend that we observe in the Fig. 4.2, shows that initially, when a single node does the whole work single handedly, it does the processing faster as it does not involve any communication and distribution delays, nor does it involve any idle time slices. As the number of nodes increases, the computation gets distributed, and, hence, is done parallely. Thus more the number of the nodes, lesser is the time taken to process a typical transaction.

4.2 Comparison with other transaction models

In tables 4.4 and 4.5 we summarize the differences between the various mobile transaction models that have been proposed so far and compare them with the *Team Transaction Model*.

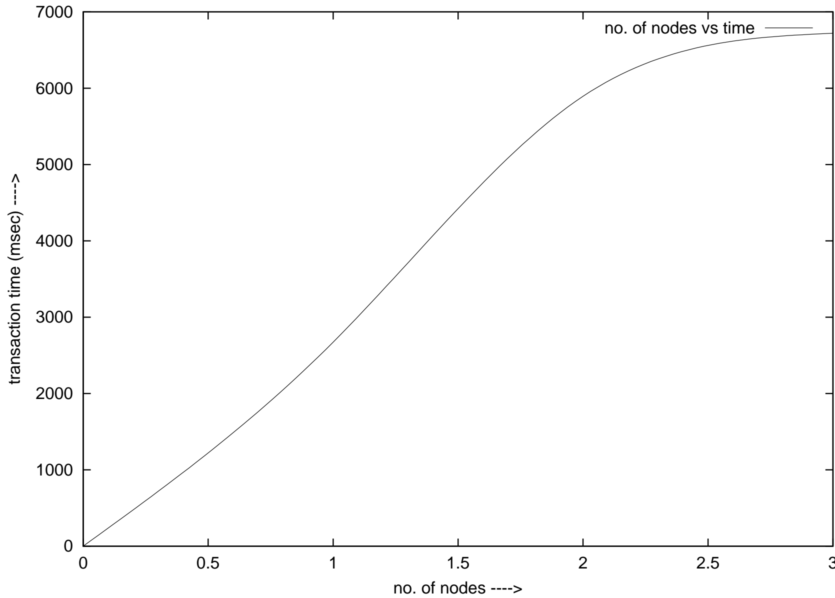


Figure 4.2: No. of nodes vs. Total Time

4.3 Conclusion

In this thesis work, we have developed and implemented a transaction processing system along with recovery techniques for mobile wireless ad hoc networks. The model uses a concept of processing transactions through a **collaborative effort** of mobile nodes. The situation becomes more challenging when the environment is purely ad hoc. The proposed system tries to adapt to all the ad hoc wireless environment issues like power constraints, scarce bandwidth availability and widely heterogeneous network.

In this work, we also discuss logging and recovery aspects related to the transactions in mobile environment. The model gives primary consideration to the long-lived nature of the transactions and the prolonged frequent disconnection periods in purely ad hoc networks. Hence in recovery, we try to restore a consistent state of the database that is closer to its state that existed at the time of crash of the coordinator. This reduces the amount of work lost in a crash, looking to the fact

Models	Consistency & Concurrency	Database system model	Additional Infrastructure
Reporting and Co-Transactions	Compensating transaction	Multi-database	Transaction Manager modified
Kangaroo model	Relies on underlying database	Heterogeneous multi-database	Data access agent
Clustering model	Bounded inter-cluster consistency	Fully distributive database	Strict and Weak transactions
Semantics based model	Based on object semantics	Distributive multi-database	Fragmentation
MDSTPM	Relies on underlying database	Heterogeneous multi-database systems	MDSTPM layer
Team Transactions	Bounded consistency guaranteed by ARIES type recovery and rollback. Can scale up to provide desired concurrency.	Heterogeneous multi-database	Data access agent

Table 4.4: Mobile Transaction Models I

that crashes (owing to power shortage and movement of a node out of the range) are more frequent in such networks.

As future work, we can explore more to find solutions that could replace the class of nodes called data access agents which are supposed to be more robust and less mobile.

Models	Net management Communication cost	User profile	Extensions for commercial DB	Scalability
Reporting and Co-Transaction	Handoff information required. Reporting Co-Transaction info exchanged.	Can be used for relocating transactions	Transaction Manager will have to be extended to handle new transaction types	Will require high bandwidth
Kangaroo model	Handoff information required. Assumes that each base station can handle transactions.	Can be used to relocate transactions		
Clustering model	Handoff information required	Used to define clusters for transaction migration	Transaction Manager should be enhanced to handle weak, strict transactions clusters definitions.	Large number of clusters or large databases could lead to cluster management
Semantics based model	Handoff information required	Not required	Objects should be fragment-able or reorder-able. Object managers will be required.	
MDSTPM	Handoff information not required. Involves only submission and querying of results.	Can be used for priority queuing	Requires the transaction Manager layer above the database system.	Transaction queuing could create a bottle neck
Team Transaction	Handoff information required.	Can be used to decompose a transaction into sub-transactions.		Can scale up as required. But affects on communication cost.

Table 4.5: Mobile Transaction Models II

Appendix A

Packet Types with Description

The mobile nodes use packets to convey information and data across themselves. A typical packet contains the following fields:

- *Destination address*
- *Source address*
- *Packet type*
- *Data*

To identify the type of message contained in the packet, *packet type* is used. All packets along with their description are listed in the table:

Packet Type	Description
REQ_DAA	Broadcasted by a coordinator/root when it needs a DAA to log its work
REP_DAA	A willing DAA replies back after creating a log for the requesting coordinator
CANCEL_DAA	Sent by a mobile node to the DAA to release its services
REQ_PLAYER	Broadcasted by a coordinator when it needs players for distributed computing
REP_PLAYER	A willing mobile node replies to the requesting coordinator

Packet Type	Description
WORK	Used by a coordinator to assign work to its player
COMMIT	Sent by a root/coordinator to its DAA to indicate the end of its (sub)transaction
DELEGATE	Sent by a player to its coordinator to denote the completion of its work
ABORT	Sent by a player to its coordinator when it aborts voluntarily
FLUSH_LOG	Used by player to periodically flush its log to its coordinator Used by coordinator and root to periodically flush its log to its DAA
ROLLBACK	Sent by a player to its coordinator when it rollbacks its work
REPLAY	Sent by a coordinator to its player re-sending the job it was given earlier
KILL	Sent by a coordinator when it wishes to kill a player transaction
NODE_CRASHED	Broadcasted by DAA to indicate the crash of a coordinator transaction
REQ_RCV_COORDY	Broadcasted by a DAA to get a coordinator for performing post recovery work of a crashed coordinator
REP_RCV_COORDY	Nodes ready to volunteer reply back to a recovery request
ACK_RCV_COORDY	A three way handshake between the DAA and the newly selected coordinator to confirm a node for the post recovery work
RCV_WORK	Sent by the DAA to the newly selected coordinator accompanying the post recovery work

Table A.1: Packet Type Description

Appendix B

Implementation Specifications

This Appendix contains the implementation specifications of the various important components of the system. The model talks about three classes of nodes - *Coordinator*, *Player* and *Data Access Agent*. The operations of each of them are mentioned in the following few pages.

B.1 Data Access Agent

The *data access agent* have been used to maintain stable logs for coordinators and use the info for recovery. the *DAA Thread* executes the following block of operations, continuously.

```
begin{DataAccessAgent}
    read packet from inQ;
    switch(packet_type)
    begin
        case REQ_DAA:
            create log for the requesting coordinator;
            reply back in affirmation;
            set AliveTimer that detects a node crash;

        case FLUSH_LOG:
```

```
                reset AliveTimer;
                append the log;

            case COMMIT:
                remove log entry of the sender node;
                remove AliveTimer;

            case CANCEL_DAA:
                remove log entry of the sender node;
                remove AliveTimer;

            case REP_RCV_COORDY:
                cancel AliveTimer;
                send ack if chosen for recovery;
        end;
    end;
```

B.2 Root/Coordinator

The *root* and a *coordinator* differ in only one respect that the root is the one that initiates a transaction. The *Job Allocator Thread* on these nodes keep executing the following set of operations, continuously.

```
begin{Root_Coordinator}
   if(new application started)
   begin
       read the JOB and DB files;
       create new application;
       assign TTID to it;
       create new log entry;
       send request for DAA;
       send request for players;
```

```
end;

read packet from inQ;
switch(packet_type)
begin
    case REP_PLAYER:
        if(distributed work available)
        begin
            assign a unique PTID;
            add it into list of pending jobs;
            set PlayerResponseTimer;
            send work;
        end;

    case REP_DAA:
        set application's DAA;
        initialize log;
        set LogFlushTimer;
        flush log to DAA periodically;

    case FLUSH_LOG:
        reset PlayerResponseTimer;
        append log;

    case ABORT:
        reset PlayerResponseTimer of the sender player;
        modify its status in list of pending jobs;
        reassign the work to other player transaction;

    case ROLLBACK:
        modify its status in list of pending jobs;
```

```
            replay the work to the player node;

        case DELEGATE:
            remove PlayerResponseTimer;
            modify its status in list of pending jobs;
            accept report;

        case NODE_CRASHED:
            cancel PlayerResponseTimer of the concerned player;

        case REQ_RCV_COORDY:
            if(not engaged in the same transaction)
                reply in affirmation;

        case ACK_RCV_COORDY:
            add TTID into list of selected TTIDs;

        case RCV_WORK:
            construct a new application;
            proceed to execute;
    end;
end;
```

B.3 Player

The player nodes are the leaf nodes, in the tree structure of a Team Transaction in progress. It keeps executing the following block of operations, continuously.

```
begin{Player}
    read packet from inQ;
    switch(packet_type)
    begin
```

```
        case REQ_PLAYER:
            if(not engaged with the same TTID)
                reply back in affirmation;

        case WORK:
            set LogFlushTimer;
            if(work can be distributed)
                 call players;
            execute operations;

        case KILL:
            remove the application for the PTID;

        case REPLAY;
            restart application;

        case REQ_RCV_COORDY:
            if(not engaged in the same transaction)
                reply in affirmation;

        case ACK_RCV_COORDY:
            add TTID into list of selected TTIDs;

        case RCV_WORK:
            construct a new application;
            proceed to execute;
    end;
end;
```

B.4 Recovery Mechanism

Whenever a coordinator crashes, its DAA detects this, and starts recovery. The details of the mechanism follows:

```
begin{RecoveryMechanism}
    broadcast node crash info;
    broadcast a request for new coordinator;
    get the log of crashed node;
    get the database items;
    get the instruction set of the node;
    analyze log to create a table of all involved (sub)transactions;
    redo updates()
      begin
        remove entries of non-committed (sub)transactions from log;
        modify the database items with the last updated value;
      end;
    using the state id's of the database items, compose the instruction
      set;
    get the id of the new coordinator;
    create a new log for the new coordinator;
    send the updated database items and modified instruction set to
      the new coordinator;
    set AliveTimer for new coordinator;
end;
```

Bibliography

[1] Chrysanthis, P. Transaction processing in mobile computing environments. In *IEEE Workshop on Advances in Parallel and Distributed Systems* (1993).

[2] Chrysanthis, P. K., and Ramamritham, K. Synthesis of extended transaction model using acta. In *Mobile Computing Environment and IEEE Workshop on Advances in Parallel and Distributed Systems* (October 1993), pp. 77–82.

[3] Dunham, M. H., Helal, A., and Balakrishnan, S. A mobile transaction that captures both data and movement behaviour. In *ACM-Baltzer Journal on Mobile Networks and Applications, Volume 2* (1997), pp. 149–162.

[4] Ghosh, R. K., Gore, M. M., Gupta, A., and Gupta, N. Team transactions: A new transaction model for mobile ad hoc networks.

[5] Gore, M. M. *Extendible, long-lived transaction processing on distributed and mobile environments with recovery guarantees.* PhD thesis, July 2001.

[6] Mohan, C., Haderle, D., Lindsat, B., Pirahesh, H., and Schwarz, P. Aries: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. In *ACM Transactions on Database Systems, Vol.17, No. 1* (March 1992), pp. 94–162.

[7] Pitoura, E., and Bhargava, B. Revisiting transaction concepts for mobile computing. In *First IEEE Workshop on Mobile Computing Systems and Applications* (June 1995), pp. 164–168.

[8] Pitoura, E., and Bhargava, B. Maintaining consistency of data in mobile distributed environments. In *15th International Conference of Distributed Computing Systems* (1996), pp. 404–413.

[9] Walborn, G., and Chrysanthis, P. Supporting semantics-based transaction processing in mobile database applications. In *14th IEEE Symposium on Reliable Distributed Systems* (1995), pp. 31–40.

[10] Walborn, G., and Chrysanthis, P. Transaction processing in pro-motion. In *1999 ACM Symposium on Applied Computing* (1999).

[11] Yeo, L., and Zaslavksy, A. Submission of transactions from workstations in a cooperative multidatabase processing environment. In *14th International Conference on Distributed Computing Systems* (1994).