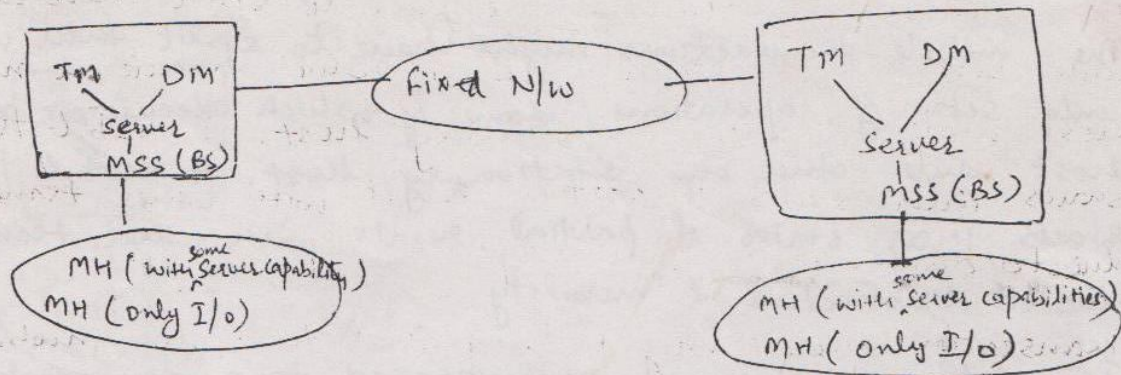


Mobile Architecture :-



MSS - Mobile Support stations also known as Base Stations (BS)

Characteristics of Mobile Transactions: → Transactions in mobile environment are different from transactions in the centralized or distributed databases in the following ways:

- (i) The mobile transactions might have to split their computations into sets of operations, some of which execute on mobile host while other on stationary host. A mobile transaction shares their states & partial result with other transactions due to disconnection & mobility.
- (ii) Transaction process may be migrated to a non-mobile computer if no further user interaction is needed.
- (iii) The mobile transactions require computations and communications to be supported by stationary hosts.
- (iv) When the mobile user continues its transaction during moves, different mechanisms are required if the user wants to continue its transaction at a new destination.
- (v) As the mobile hosts move from one cell to another, the states of transaction, states of accessed data objects, and the location information also moves.
- (vi) The mobile transactions are long-lived transactions due to the mobility of both the data and users, and due to frequent disconnections.
- (vii) The mobile transactions support and handle concurrency, recovery, disconnection and mutual consistency of the replicated data objects.
- (viii) The execution of transaction is fully coordinated by system in distributed computing whereas the movement of mobile unit controls the execution of mobile transaction.

Mobile Transaction Models :-

- Semantic based transaction processing models have been extended for mobile computing to increase concurrency by exploiting commutative operations. These techniques require caching large portions of the database or maintain multiple copies of many data items. Fragmentability of data objects have been used to facilitate semantic based transaction processing in mobile databases. The schemes fragments data object & each fragmented data object has to be cached independently and manipulated synchronously. This scheme works in the situations where the data objects can be fragmented like stacks & queues.
- In optimistic concurrency based schemes, cached objects on mobile hosts can be updated without any coordination but the updates needs to be propagated and validated at the database servers for the commitment of transactions. This schemes leads to aborts of mobile transactions unless the conflicts are rare. Since mobile transactions are expected to be long lived due to disconnection & long network delays, the conflicts will be more in mobile computing environment.
- In pessimistic schemes in which cached objects can be locked exclusively, mobile transaction can be committed locally. The pessimistic schemes leads to unnecessary transaction blocking since mobile host can not release any cached objects while it is disconnected. Existing caching methods attempt to cache

the entire data objects or in some case the complete file. Copying of these potentially larger objects over low bandwidth communication channels can result in wireless network congestion and high communication cost. The limited memory size of the MHA allows only a small number of objects can be cached at any time.

→ The concept of transaction proxies is introduced to support recovery. For each transaction submitted at a MHA, a dual transaction includes the updates of the original transaction. Proxy transaction takes the periodic backup of the computation performed at mobile host.

→ The notion of twin-transaction was introduced which essentially replicates the process of executing transactions. In twin-transaction model, each write request will be mirrored and two equivalent transactions will be created. If a mobile host is disconnected, the transaction execution can still be proceed. It is an improvement to transaction proxies where disconnections were not considered in detail.

The proposed system is designed to support the execution of transactions in a distributed environment. The system is designed to support the execution of transactions in a distributed environment. The system is designed to support the execution of transactions in a distributed environment.

Cluster Model:- This model is a flexible, two-level consistency model which deals with the frequent, predictable and varying disconnections. The model is based on grouping semantically related or closely located data together to form a cluster. Data are stored or cached at a mobile host (MH) to support its autonomous operations during disconnections. Transaction in this model may involve both remote data and data stored locally at the user's device.

The items of a database are partitioned into clusters and they are the units of consistency in that all data items inside a cluster are required to be fully consistent, while data items residing at different clusters may exhibit bounded inconsistencies.

Clustering may be constructed depending on the physical location of data. By using this locality definition, data located at the same, neighbor, or strongly connected hosts may be considered to belong to same cluster, while data residing at disconnected or long distance locations may be regarded as belonging to separate clusters. In this way, a dynamic cluster configuration will be created when a user enters into a new cell, it can change its cluster too. Therefore, clusters of data may be explicitly created or merged by a probable disconnection or connection of the associated mobile host.

Clusters of data may be defined by using the semantic of data such as the location data or by defining user profiles. Location data, which represent the address of a mobile host, are fast changing ~~replicated over many sites~~ data because of the uncertain location of mobile host and so needs to be replicated over many sites again and again. Since updating all the copies creates overhead and there may be no need to provide consistency for these kind of data.

By defining user profiles for the cluster creation, it may be possible to differentiate users based on the requirements of their data and applications. For example, data that are most often accessed by some user or data that are somewhat private to user can be considered to belong to the same cluster independent of their location or semantics.

Integrity constraints specify or ~~express~~ the relationship of data items that a database state must satisfy in the form of restrictions. Integrity constraints among data items inside the same cluster are called intra-cluster constraints and constraints among data items at different clusters are called inter-cluster constraints. During disconnections user may not be able to satisfy inter-cluster constraints strictly.

- Two basic types of transaction are defined in cluster model.
- (i) Weak transaction: - consists only weak read & weak write operations and they only access data copies that belong to same cluster and can be considered local at that cluster.
- (ii) Strict transaction: - standard read & write operations are called strict read & strict write operation which forms a strict transaction.

A weak read operation on a data item reads a locally available copy, which is the value written by last weak or strict write operation at that cluster. A weak write operation writes a local copy and is not permanent unless it is committed in the merged network.

A strict read operation is defined as the one that reads the value of the data item which is written by the last strict write operation where a strict write operation means writing one or more copies of the data item.

Weak transactions have two commit points, a local commit in the associated cluster and an implicit global commit after cluster merging. Updates made by locally committed weak transactions are only visible to other weak transactions in the same cluster, but not visible to strict transactions before merging, or local transactions become globally committed.

Weak operations support disconnected operation since a mobile device can operate disconnected as long as applications are satisfied with local copies. Users can use weak transaction to update mostly private data and strict transactions to update highly used common data. Furthermore, by allowing applications to specify their consistency requirements, better bandwidth utilization can be achieved.

Multi database Transactions:- The mobile host can play many roles in a distributed database environment. It may simply submit operations to be executed on a server or an agent at a fixed network. Each mobile client is assumed to submit a transaction to a coordinating agent. Once the transaction has been submitted, the co-ordinating agent schedules and co-ordinates its execution on behalf of the mobile client. Mobile units may voluntarily disconnect from the network prior to having any associated transactions completed.

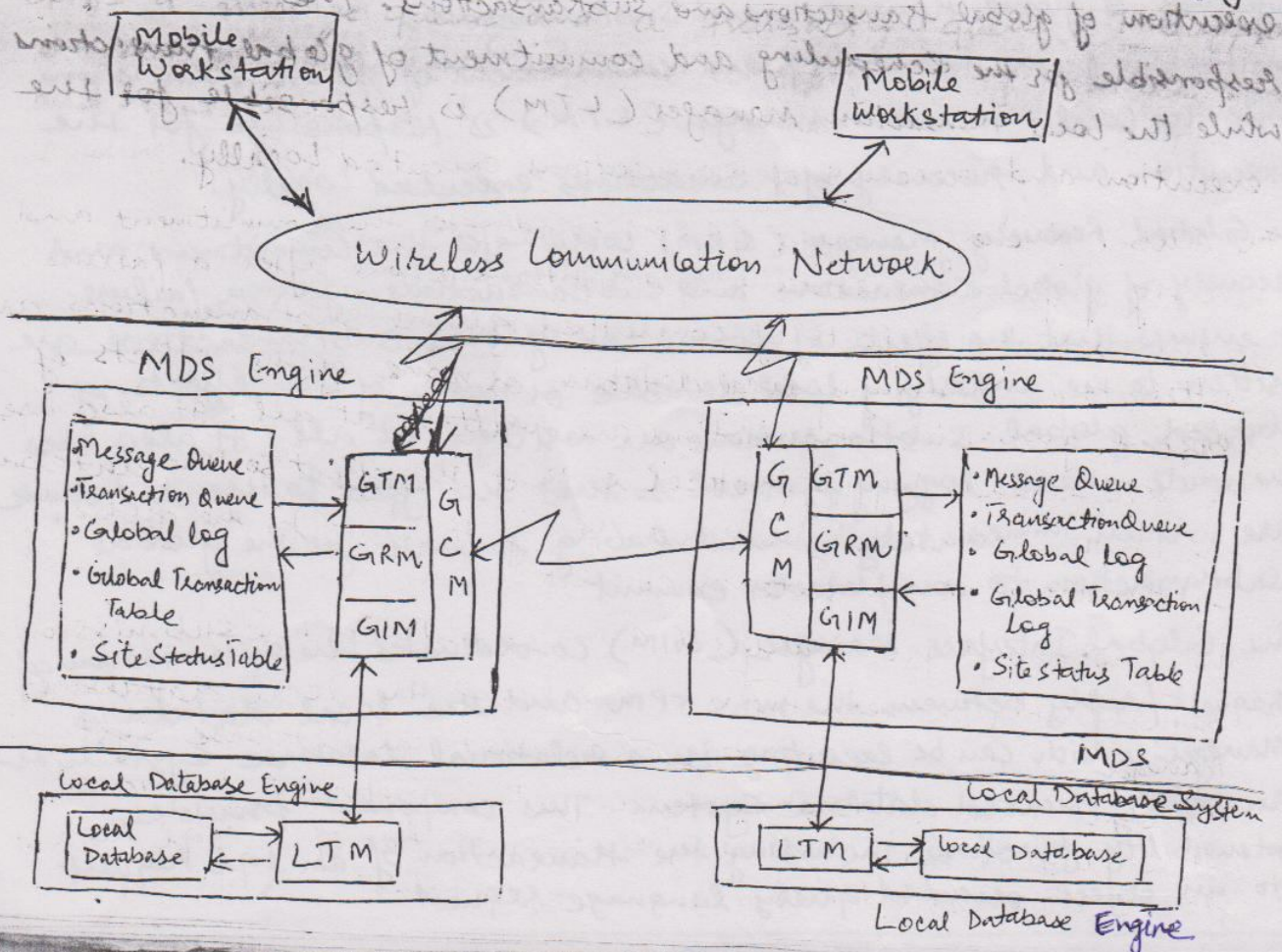
They aimed at architecture that satisfies the following:

- Providing full-fledged transaction management framework so that the users and applications will be able to access data across multiple sites transparently
- Enhancing database concurrency and data availability through the adoption of a distributed concurrency control and recovery mechanism that preserves local autonomy
- implementing the concept extensibility to support various database systems in the framework so that the components can co-operate with a relational or an object oriented database system.
- Providing an environment where the proposed transaction processing component operates independently and transparently of the local DBMS
- incorporating the concept of mobile computing through the use of mobile workstations into the model.

MDSTPM System Architecture:

(5)

A multidatabase system (MDS) is defined as an integrated distributed database system consisting of a number of autonomous component database management systems. Each of the underlying component database system is responsible for the management of transactions locally. To facilitate the execution of global transactions, an additional layer of S/W must be implemented which permits the scheduling and co-ordination of transactions across these heterogeneous database management systems. The proposed Multidatabase Transaction Processing Manager (MDSTPM) architecture combining mobile computing is shown in figure.



The MDS TPM consists of the following components

- The Global Communication Manager (GCM) is responsible for generation and management of message Queues within the local site. Additionally, it also communicates, delivers and exchanges these messages with its peer sites and mobile hosts in the network.
- The Global Transaction Manager (GTM) coordinates the submission of global Subtransactions to its relevant sites. The Global Transaction Manager Coordinator (GTMC) is the site where the global transaction is initiated. All participating GTMs for that global transaction are known as Global transaction manager Processors (GTMPs). The GTM can be a Global scheduling Submanager (GSS) or a Global Concurrency Submanager (GCS). The GCS is responsible for acquisition of necessary concurrency control techniques needed for the successful execution of global transactions and subtransactions. The GTM is responsible for the scheduling and commitment of global transactions while the local transaction manager (LTM) is responsible for the execution and recovery of transactions executed locally.
- The Global Recovery Manager (GRM) coordinates the Commitment and recovery of global transactions and subtransactions after a failure. It ensures that the effects of committed global subtransactions are written to the underlying local database or none of the effects of aborted global subtransactions are written at all. It also uses the write-ahead logging protocol so that the effect to the database are written immediately without having to wait for the global subtransaction to complete or commit.
- The Global Interface Manager (GIM) co-ordinates the submission of request/reply between the MDS TPM and the local database Manager which can be executing in a relational database system or an object-oriented database system. This component provides extensibility function including the translation of an sql request to an object-oriented query language request.

In this approach, the mobile workstations need to ~~be a~~ ^{be a} part of MDS ^⑥ using its connection with their respective co-ordinator node. On global transaction submission, the coordinating site can then schedule and coordinate the execution of the global transaction on behalf of the mobile host. In this way, the mobile workstation may disconnect from the network without waiting the global transaction to complete.

An Alternative mechanism to RPC (Remote procedure call) is proposed as message and Queuing Facility (MQF) for the implementation of the proposed approach. Messages are handled asynchronously by coordinating site providing the mobile host to disconnect. Also the mobile hosts can query the status of global transaction.

In the proposed MQF, for each mobile workstation there exists a message queue and a transaction queue. This approach is based on finite state machine concept. Set of possible state and transactions can be clearly defined between the beginning and ending state of global transaction. Five transaction sub queues are used

→ input queue → allocate queue → active queue
→ suspended queue → output queue.

It is necessary to ~~provide~~ establish an MDSTPM component s/w at each site in order to provide integration.

This model ignores important issues including interactive transactions that need input from the user and produce output, transactions that involve data stored at mobile workstations and mobile host migration.

KANGAROO TRANSACTIONS

The reference model consists of three layers:-

- (i) The Source System: The Source system represents a collection of registered systems that offer information services to mobile users.
- (ii) The Data Access Agent: Data in the source system(s) is accessed by the mobile transaction through the Data access agent (DAA). Each base station hosts a DAA.
- (iii) The Mobile Transaction: When DAA receives a transaction request from a mobile user, that transaction is called a mobile transaction. mobile transaction is ~~identified~~ defined as the basic unit of computation in the mobile environment and is identified by the collection of sites (base stations) in which it hops through.

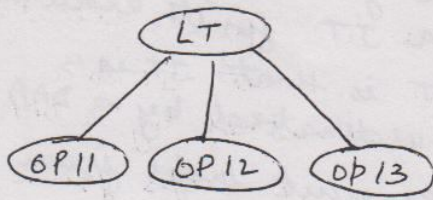
DAA on receiving transaction from mobile user forwards it to specific base station or fixed host which contains the needed data and source system component. When the mobile user is handed over to another base station, the DAA at the new station receives transaction information from old base station. DAA performs the management of the mobile transaction & this component of DAA is called the Mobile Transaction Manager (MTM) and data access coordinator for the site. It is built on the top of an existing Global Database System (GDBS). A GDBS assumes that the local DBMS system performs the required transaction processing functions including recovery & concurrency. GDBS is not aware of the mobile nature of some nodes in the network.

When a mobile transaction moves to a new cell, the control of the transaction may move or may remain at the originating site. If it remains at the originating site, message would have to be sent from the originating site to the current base station any time the mobile unit requests information. If the transaction management function moves with the mobile unit, the overhead of these messages can be avoided. For the logging side of this movement, each DAA will have the log information for its corresponding portion of the executed transaction.

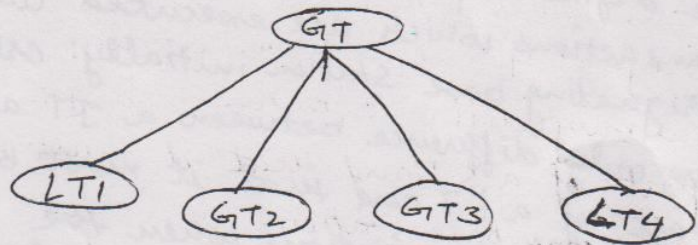
This model is built on traditional transactions which are a sequence of operations executed under the control of one DBMS. Figure 1 shows the basic structure. Here three operations (op11, op12 and op13) are performed as part of the transaction. LT is used to identify the traditional transaction as it is executed as 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.

But our view is somewhat broader than that which is often assumed. We actually consider two types of global transactions. The limited view of global transaction (G_{LT}) is shown in figure 2. Notice that the global transaction 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 Subtransactions, G_{ST}) of the G_{LT}. Each of these can in turn be viewed as consisting of sequence of operation. Figure 3 takes the more general view of G_{LT}. In this case the subtransactions may themselves be global transaction to another multi-database system. So we

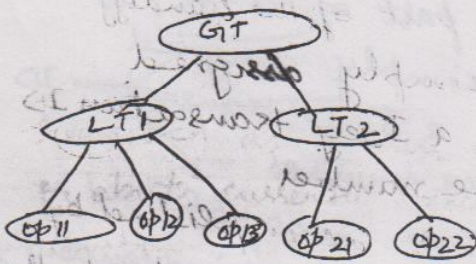
would have operations underneath LT1 and LT4. Underneath GT2 and GT3 would be other LTS and GTS. This view of global transactions gives a recursive definition based on the limiting bottom view of local transactions.



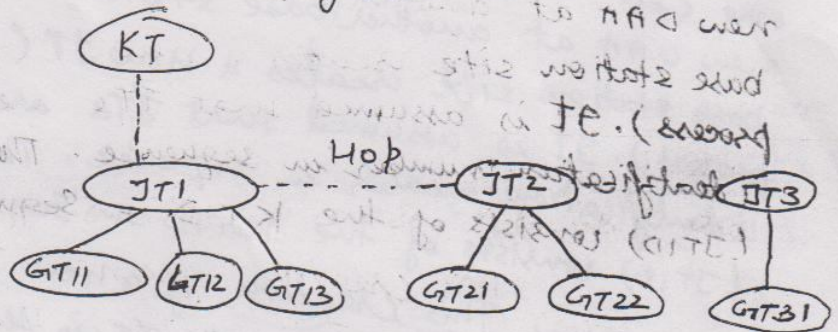
Fig(1)



Fig(3)



Fig(2)



Fig(4)

Global transactions serve as basis upon which mobile transactions are defined. Global transactions alone, however, do not capture the hopping nature of mobile transactions. Based on hopping property, we call our model of mobile transactions Kangaroo transactions (KT). Figure (4) shows the basic structure of a Kangaroo transaction. When a transaction 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. We define a KTID as

$$KTID = \text{Base Station ID} + \text{Sequence Number}$$

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). We define a pouch to be the sequence of global and local transactions which are executed under a given KT. The originating base station initially creates a JT for its execution.

The only difference between a JT and GT is that JT is a part of a KT and that it must be co-ordinated by a DAA at some base station. 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 new JT. In figure 4, 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 DBMS must be assured.

To manage a KT execution and recovery, a doubly linked list is maintained between the base station sites involved in executing a KT. 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 a transaction which is interrupted, the user must be able to provide the starting site. To undo a KT, the linked list is traversed in

backward manner.

There are two different modes for kangaroo Transactions. ④ ⑤

- i) Compensating mode
- ii) split mode.

In compensating mode, the failure of any JT causes the current JT and any preceding ~~of~~ following JTs to be undone.

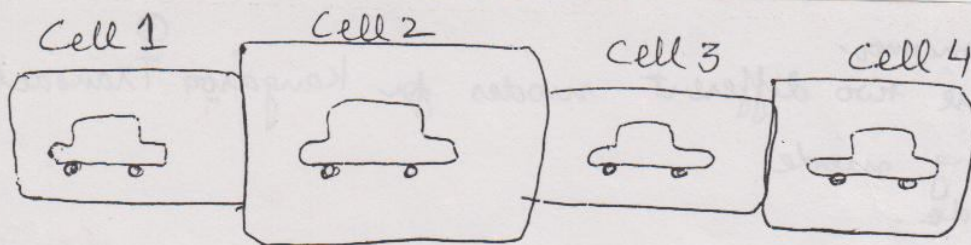
Previously committed JTs will have to be compensated for & source system or user must provide information for compensating the transaction.

Deciding ^{at first place} that a JT is compensatable or not is a difficult task. The source system should also be able to guarantee the successful commitment of compensating transaction.

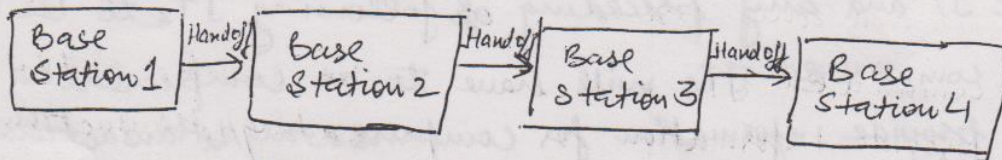
The split mode is the default mode. In this mode, when a JT fails no new global or local transactions are requested as a part of KT. However, the decision as to commit or abort currently executing ones is, of course, left up to the component DBMS. Previously committed JTs will not be compensated for.

Neither the compensating nor split modes guarantees serializability of the kangaroo transactions. Although compensating mode ensures serializability of JTs. Compensating mode ensures the atomicity of KT, but isolation may be violated (thus violating the ACID principle for transaction)

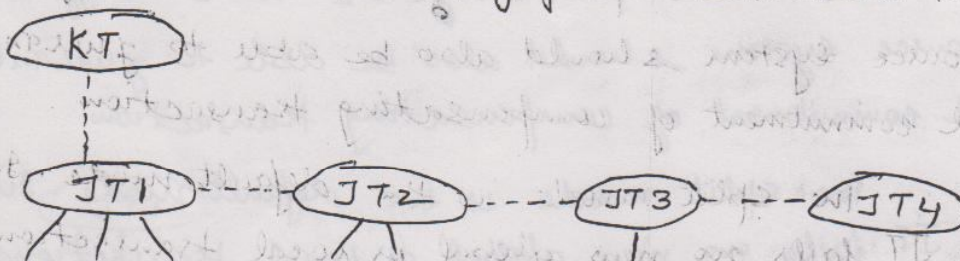
Figure 5 shows ~~movement~~ relationship between movement of mobile unit between cells and corresponding KT. Assuming that when the transaction initiated, the mobile unit is in cell 1 which is associated with Base station 1. The DAA at this base station created a new KT and immediately created JT. When the mobile unit moves to cell 2, a handoff is performed.



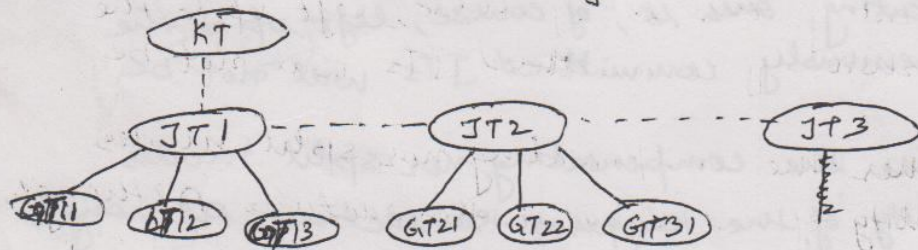
5(a) Movement of Mobile unit through cells



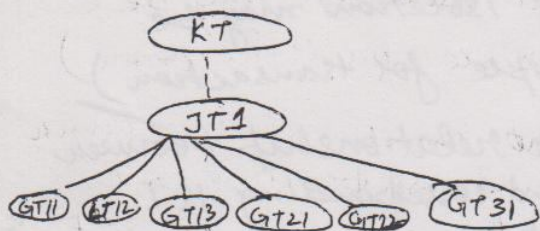
5(b) Hopping from Base Station to Base Station



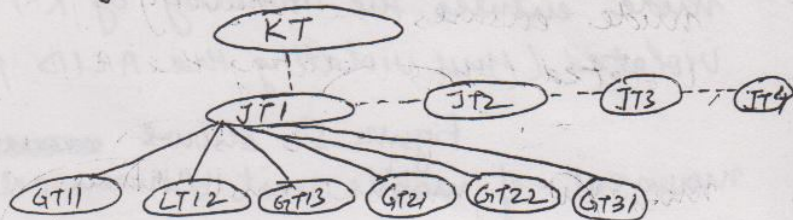
5(c) Kangaroo Transaction



6(a) One possible KT for adjustment Transaction in fig(4)



6(b) An equivalent KT



6(c) Another equivalent KT

Starting site. To undo a KT, the linked list is

(10) (7)

As a part of this handoff, the KT is split into two transactions. The part of this transaction is the subtransactions under JT1, the remaining part will be under JT2. Similarly when the mobile unit moves into cell 3 and cell 4, JT3 & JT4 are created via split operation as part of handoff. Note that this process is dynamic. A new Joey is created only when a drop between cell occurs: no drop - no Joey.

The same transaction requested at two different times could have different structures. We can illustrate this fact from fig 6. Figure 6(b) represents a short lived or slow mobility type of transaction, whereas figure 6(c) shows a long lived or fast mobility transaction. This figure 6 shows three different transactions which are equivalent to that in fig (4). Structure of each is different in terms of Joey, the underlying set of global & local transactions of each is the same. Thus, we can say that two Kangaroo Transactions are Equivalent if they have same Pouch.

We assume that the subtransactions of the mobile transaction are executed in sequence. Thus the local and global transactions under JT1 will all occur before those in JT2. This guarantees that JT1 precedes JT2 in serializable order. We assume that no subtransactions under JT2 will be created until the currently executing one in JT1 is committed. The JT2, however, must be created when handoff occurs. This situation is shown in fig 5(c) where JT4 has been created but no subtransaction yet exists.

Our discussion above assumes that subtransactions are requested by the user at the MU, but they may actually be requested by an agent at the base station.

Kangaroo Transaction Processing -

The flow of control of processing Kangaroo Transaction 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 the communication cell. At the end of this setup, a BTKT (Begin Trans. KT) record is written into the MTM transaction log.
- 2) Creating of JT by MTM also involves in generating a unique JTID and creating an entry in the transaction status table. The count of number of active Joey in KT status table entry is incremented by 1. A Begin Transaction JT (BTJT) record is then written into the log. Finally a JT entry is written into the transaction status table.
- 3) The Kangaroo transaction is executed in the context of JT. For each JT, transaction is made to map the KT operations into specific source system global & local transactions. Based on the transaction results, an ST entry is created in the transaction table for each native (local or global) transaction. A log record, BTST, is written in the MTM log before shipping each native transaction to the appropriate source system DBMS or G.DBs. Subtransactions within a JT have begin (BTST) and end (ETST) records.

Starting site. To undo a KT, the

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 the DAA writes an HOKT (Handoff KT) record in the log indicate that a handoff has taken place. In addition, the log to indicate that a handoff has taken place. The log buffer is flushed to stable log on disk. These actions represent a checkpoint operation. Notice that Subtransactions may still be executing at the source system sites.

5) The destination base station then initiates the other part of the split operation. A CTKT record is logged after a new JT is created with the tentative content being the rest of the entire KT. In addition, a KT entry is placed in the destination's status table.

6) To ensure the atomicity of the source system DBMS and GDBS systems, they will determine when to commit or abort the subtransaction. When the DAA is notified that subtransaction has been committed, the DAA writes ETST record in the log. If the KT processing 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 of KT is later aborted and the compensating transaction must be executed. The ST list in the status table is updated to reflect the fact that this subtransaction has committed. If there is no active subtransaction for Joey, then ETST record is written to the log and the status of the JT status table entry is changed to committed. Finally the count for the active Joys is decremented by 1. Note that the KT status table entry for the last active base station contains the current values of for status and Joey count. If the Joey count is 0 and the status of KT is committing, then 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~~ At this time, if the active log count is 0, the KT is committed.

8) To commit a KT, the ETKT entry is written to the log and all status table entries for all involved base stations are freed.

Logging & Recovery with Kangaroo Transactions

To illustrate recovery possibilities, in given table we show a sequence of actions and log records created for a simple transaction.

Action	Log records at BS1	Log records at BS2	Comments
Transaction requested at MU	(BTKT, BS1.1, Split)		Create KT, JT, ST
	(BTJT, BS1.1.1, Λ)		
	(BTST, BS1.1.1.1, Reg, Λ)		
Handoff	(HOKT, BS1.2)	(CTKT, BS1.2, Split)	Create JT
	(BTJT, BS1.2, BS1.1.1)		
BS1 commits	(ETST, BS1.1.1.1)		
Subtransaction 2	(ETJT, BS1.2.1)		Commit JT, ST
Another subtransaction requested	(BTST, BS1.2.1, Reg, Λ)		Create ST

The MU requested a transaction through base station 1 (BS1). As a result a KT (BS1.1), a JT (BS1.1.1), and ST (BS1.1.1.1) are created. The Λ indicates an empty value. The MU then moves into base station 2 (BS2). As a result an HOKT is written into the log at BS1 and a CTKT is written at BS2. Immediately a new JT (BS1.2) is created and a corresponding log record is written. Even though the split has occurred, the previous JT & ST can't be committed until the message is received from BS1. At that time ET records are written into log & next ST created. Suppose an MU failure occurs (or it is lost or stolen) at this point in time. Since the KT is of split mode, the currently executing JT & ST will be allocated to complete, but no new STs will be created. No real recovery occurs. This allows the transaction to be restarted at a later point if desired. If KT is of compensating mode, then a compensating transaction would have been created for each subtransaction (BS1.1.1.1 & BS1.2.1). So work of these subtransactions (as a result the KT) will be undone.

starting state.