Distributed Algorithms for Mobile Environment

R. K. Ghosh

May, 2005

5.1 Introduction

From application prospectives, a mobile computing system is a distributed systems consisting of one more mobile devices and a set of static computers connected by a computer network. A major part of the research in mobile computing system is directed towards establishing and maintaining connectivity between mobile terminals with static computers through bridges between wireless with wired networks. Over the years, however, mobile computing has emerged as a distinct paradigm with problems that are characteristically different from those in normal distributed computing.

From an abstract point of view, a mobile computing system can be seen as graph that consists of a fixed core of static nodes and a dynamic set of mobile leaf nodes [6]. It is much like a cellular mobile telephone network. The mobile leaf nodes can be viewed as a set of persistent messages moving through graph. With this underlying graph model, traditional distributed algorithms can be implemented directly on mobile system. Unfortunately, a direct mapping of distributed algorithms to mobile environment is not practical due to limited bandwidth, fragility of wireless links and many other constraints connected with mobile nodes.

A commonsense driven approach to design a distributed algorithm for mobile system will be to leverage as much as possible the fixed part. It intrinsically formulates a logical two-tier approach to computing in a mobile distributed environment. By offloading most of the compute intensive tasks on the computers belonging to the fixed network, mobile devices can save its critical resources including battery power. Before we expand on the stated idea of two-tier approach, let us examine the difference between a traditional distributed system and a mobile computing system.

The mobility of a computing node brings up two important new issues in data delivery:

1. Locating a node for delivery of message, and
2. Transparent semantic routing of the message thereafter.

Consequently, any attempt to map existing distributed algorithms for execution in mobile computing environment in a simple way will unlikely to meet with much success. Nevertheless, observing the differences between mobile computing and distributed systems will help in recognizing the issues that may arise in design of distributed algorithms or restructuring existing distributed algorithms for execution on mobile computing systems.

5.2 Distributed Systems and Algorithms

A distributed system consists of a set of autonomous computers (nodes) which communicate through a wired network. A program which runs on distributed system is typically organized as a collection of processes distributed over different nodes of a distributed system. The computation performed such a program is called a distributed computation. A distributed computation consists of four iterative steps, (i) broadcasting, (ii) information gathering, (iii) executing some joint computation, and (iv) agreeing on coordinated actions by the participating processes. The last three steps are closely related. Information gathering in a distributed setup requires participants to exchange message amongst themselves. Similarly joint computation progresses with exchange partial results amongst participants. Likewise, a coordinated action requires information exchange. Since the processes are distributed over a set of autonomous computers, synchronization requirements a distributed algorithm can be met by either through a shared memory or through exchange of messages over the network among the nodes. A shared memory in a distributed system is mostly implemented at software level either transparently by extending the underlying virtual memory architecture or by explicitly through a set of library functions. So, typically message passing is the basic interface of exchanging information between compute nodes in a distributed system.

All distributed algorithms are designed with certain basic assumptions about the capabilities of the participating nodes.

1. The nodes are static and their locations (IP/MAC addresses) are known in advance. Therefore, no cost is incurred for locating a host.

2. The message setup cost is fixed and same for all the messages. Therefore, the transfer of message dominates messaging cost.

3. Sufficient bandwidth is available, transfer of large messages possibly may incur long network latencies. Therefore, it suffices to assume that a fixed cost is incurred for sending a message between two fixed nodes.

4. The participating are nodes resource rich, having enough computation power, memory.

5. The nodes are powered by continuous supply of power. Hence, do not have energy problem, and remain active during the full execution of

the algorithm.

6. Their inability to receive a message due to a power failure is treated as a failure of the algorithm.

5.3 Mobile Systems and Algorithms

Before dealing with the design of distributed algorithms for mobile environment, there is need to understand how at all such algorithms can be evaluated to determine their efficiencies, because the evaluation criteria influence the design of algorithms. As obvious from the characteristics of a mobile computing environment, the efficiency requirement of a distributed algorithm suited for such an environment should be to

- Minimize communication cost,
- Minimize bandwidth requirement,
- Meet the synchronization requirements, and
- Overcome resource constraints of mobile hosts.

Bandwidth is usually treated as a resource. Therefore, the impact of poor bandwidth can be examined along with the other resource constraints.

Synchronization is a key issue for the correct execution of any distributed algorithm. Unlike static clients, mobile clients can appear and disappear in any cell in a service area at any time. So, synchronization techniques should be adjusted to handle dynamically changing locations of the peers. Therefore, synchronization of mobile clients more complicated and far more expensive than that of clients in a static environment. There is, thus, a need to evolve of a new model for evaluating the cost of distributed algorithms in mobile environments. Apart from some easily identifiable cost criteria such as,

- Computation on a mobile node versus that on a static node,
- Relocating computation to static host, and
- Communication on wireless channels.

There are other cost related elements such as *disconnection* of a mobile host from the fixed network and its subsequent reconnection to the fixed network. Furthermore, the cost model applicable for mobile infrastructured network is not directly extendible to infrastructureless mobile ad hoc networks. So, separate cost models need to be developed for different mobile networks.

The above discussions indicate that the efficiency of algorithms for mobile environment can evaluated under following three broad criteria:

- How best a compution in a mobile environment can be modeled?
- How synchronization and contention problems can be resolved?
- How a cost model for messaging can be developed?

5.3.1 Placing computation

Whenever an operation is to be executed on a remote object it is achieved by sending messages to the node that hosts that object. The operation then gets performed by the remote node on behalf of the initiating host. It is convenient to think that the mobile host *logically* executing operations directly on the remote node. Sending a message to a mobile host itself is two step task:

1. First step is to locate the mobile host.
2. Next step is to actually send the message.

If the message is meant for a fixed host, the above two steps can be carried out by the base station (BS) of the mobile host within the fixed network. The BS being a part of fixed network would be able to forward the message to the destination node by leveraging the IP forwarding protocol. Sending messages to a fixed host is thus, lot less expensive than sending messages to a mobile host, because no location search is required for delivering messages to a fixed host. Thus, it is preferable to avoid sending messages to mobile hosts except the case when both sender and the receiver are under the same BS. As mobile hosts are required to avoid sending messages to one another, a mobile host should avoid executing operations on objects resident in another mobile host. So, the first design principle is:

Principle 1 *[1] To the extent possible, all remotely accessed objects should be resident on the fixed hosts.*

In our case an object is associated with the entity that requires computing power and bandwidth, thus, we disallow such things on the mobile host. The role of the mobile host comes up, as being involved in the thread of execution which executes the operations on the object (in most case the operation is staged by sending message to the fixed remote node holding the object).

5.3.2 Synchronization and contention

Whenever a particular resource is accessed from a number of remote locations there would be contention. We can treat this resource as an object. Each sequence of operations that are being initiated from a specific place (a mobile host) can be called a *thread* of execution. Thus, execution scenario is that of many concurrently running threads trying to operate on an object. For the moment, let us not make any assumptions about where this object is located. It may be either be resident on a fixed host or on a mobile host. The concurrent threads compete to gain access to the object. It is undesirable to have the object in an inconsistent state due to concurrent operations by the competing threads. In other words, the access to on the object by mutually competing threads should be synchronized.

Let us examine why mutual exclusion is an important issue in a distributed settings. The general pattern of all distributed algorithms can be visualised as comprising a *communication* component and *computation* component. The computation component is limited to the individual hosts and during execution of a computation, a host might need to communicate with its neighbours or other nodes for exchanging the results of partial computations. So further progress in computation may need synchronization. The synchronization requirements among other things may involve parameter initializations for the next phase of computation. Thus a distributed system of hosts exhibit bursts of communication in between periods of local computation.

When hosts become mobile, one additional cost parameters, namely, *cost of location lookup* is introduced. Furthermore, due to resource poorness of mobile hosts, the cost assignment criteria of computational resources at mobile hosts become drastically different from that at fixed host. The communication cost also needs to distinguish between costs incurred for messaging over wired and wireless links. A simple technique, to avoid the high resource cost at mobile hosts is to relocate compute intensive parts of the algorithm, as much as possible, to the fixed hosts. To address location lookup cost,

three different strategies, namely, the *search*, *inform* and *proxy* are proposed by Badrinath, Acharya and Imielinski [1] in the context of restructuring a logical token ring network. These techniques are generic in nature and, therefore, can be used in various other classes of distributed algorithms. We will examine these strategies

5.3.3 Messaging cost

One of the important complexity measures of distributed algorithms is the communication cost. It is dependent on the number of messages exchanged during one execution of the algorithm. But when hosts become mobile, the communication complexity should also include the cost of location search. Location search includes the messages exchanged to find the location of a mobile host. We know that mobile hosts have severe power constraints. They can send messages only on wireless links which is the major source of power drainage. So, the communication cost over a wireless link is more costly than that over the conventional wired link. Badrinath, Acharya and Imielinski [1] proposed three different measures of cost for counting number of messages exchanged during execution of a distributed algorithms in a mobile computing environment. The cost model is as follows:

- C_w: cost of sending a message from MH to BS over wireless channel (and vice versa)
- C_f: cost of sending a message from a point to another point linked by the wired N/W.
- C_s: cost of searching/locating the current base station BS_{cur} of an MH and forwarding a message from a source base station BS_{src} to BS_{cur}.

The simplest strategy to locate a mobile host is to let the searching base station query all other base stations and determine if a mobile host belongs to a cell under one of the set of queried base stations. The base station responding to the query is the one which has the mobile host in the cell under it. The querying base station can then forward the message meant for the mobile host to the responding base station. So, message exchanged for location search as illustrated by Figure 5.1 will be as under.

1. In the first round all base stations except one receive message from the querying base station. So total of $(N_{BS} - 1) \times C_f$ messages are exchanged.

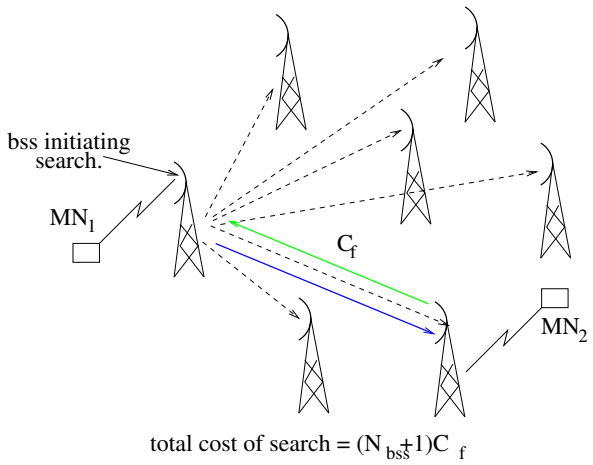

Figure 5.1: Search cost

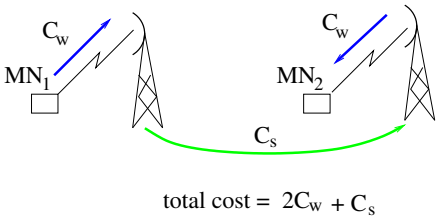

Figure 5.2: Mobile to mobile communication.

2. The base station having the searched mobile host responds. This incurs a cost of C_f.

3. Finally querying base station forwards a message (data packet) to the responding base station. This incurs a cost of C_f.

Adding all three costs, the worst case cost for search is equal to $(N_{BS}+1)\times C_f$. Figure 5.2 explains how a message from a source mobile can be delivered to a destination mobile. counts when The cost sending message between the sender (MH) and the receiver (MH′) is determined by overhead plus the cost of actual message transfer. The break down of the cost is given below.

1. The source MH sends the message to its own base station BS. The cost incurred is C_w

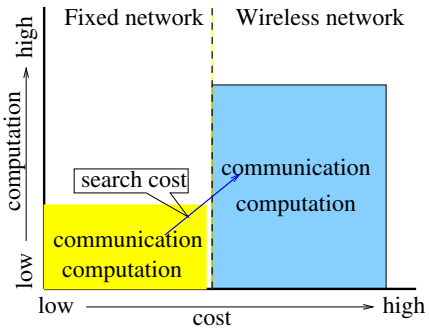

Figure 5.3: Summary of cost model.

2. BS then searches for the destination MH′ to find the base station BS′ under whose cell area MH′ is currently located and deliver the message to BS′. The cost incurred for this action is C_s.

3. BS′ sends the data packet to final destination MH′. An action incurring a cost of C_w.

Adding the costs together, the worst case cost of sending a message from a mobile host MH to another mobile host MH′ can be at most $2C_w + C_s$.

The analysis of the cost structure which Badrinath, Acharya and Imielinski [1] have proposed is captured succinctly in Figure 5.3. The major component of the cost in a mobile distributed environment is due to communication. Computation is also slow in mobile hosts, so, in the cost due to computation is relatively high compared that in conventional distributed system. However, communication between a fixed host and a mobile host may lead to a substantial cost in fixed network mainly due to the fact a search is needed to locate the mobile host.

5.4 Structuring Distributed Algorithms

The attempts to execute distributed algorithms directly without any structuring on mobile environment leads to design of inefficient algorithms. The inefficiencies, in design of algorithms for mobile distributed environment, as observed in the previous section arise out of synchronization, asymmetric model of commputation, and imbalance in communication cost between

wired and wireless interfaces. From the point of view of algorithm design the problems are dependent on the abstract notion of coordination and control exercised in distributed algorithms. As an analogy, consider the issue of quality in software design process. The efficiency of algorithms addresses the quality in the problem domain. But when algorithms are converted to software processes, the issue of quality goes beyond the problem domain; it becomes linked to the choice of technologies for implementation such as computers and their capabilities, underlying network, storage, programming tools and languages, etc.

In a distributed computing environment, the control is not exercised by a single computer. So the efficiency issue, i.e., the issue of quality must consider how control and coordination are exercised in the execution of distributed algorithms. Therefore, in order to structure distributed algorithm for execution in mobile environment we also need to consider the problem of coordination and control. The class of distributed systems can categorized as follows.

- Non-coordinator based systems:
- Coordinator based systems:

5.5 Non-Coordinator Systems

In a non-coordinator based system, all machines are equivalent in the sense no machine exercises any control on another machine. A non-coordinator system is known more popularly as a peer-to-peer system. There are two different types of non-coordinator based systems. In the first type of non-coordiator based system, each peer execute the same code. In the second type, a few of the machines execute some specialized code, while the rest excute the same code.

5.5.1 All Machines are Equivalent

In such a system the amount of computational and communication load shared by each machine is roughly the same. Such systems can easily be modified to work in a mobile computing environment. We illustrate this with Lamport's Bakery algorithm [**?**] for mutual exclusion. In the Bakery algorithm, a process waiting to enter critical section chooses a number. It

allows all processes which have chosen smaller numbers to enter into critical section before itself. Ties are resolved by process ids allowing the process with lower id to enter critical section. Lamport's original algorithm makes use of two shared arrays each consisting of n elements. One element is assigned for each process in each shared array. The values in shared array can be examined by any process. In a peer to peer settings, no shared arrays can be used. So, local variables `choose` and `value` are maintained by each process. A process uses message passing mechanism when it needs to examine the values of local variables of another process. The pseudo code of the algorithm appears below.

boolean choosing$_i$ = false;
int number$_i$ = 0;
while (1) {
    choosing$_i$ = true;
    set value$_i$ = max {value$_j | j \neq i, j = 0 \ldots N_{MH} - 1$} + 1;
    choosing$_i$ = false;
    for ($j = 0$; $j < N_{MH}$, j != i; $j++$) {
        while(choosing$_j$);
        while(number$_j$!= 0) && ((number$_j$, j) < (number$_i$, i));
    }
    ...
    { Critical section code}
    ...
    number$_i$ = 0;
}

To appreciate the impact of executing Lamport's Bakery algorithm directly on the mobile hosts one need not look into the details of its execution, but just focus on the communication aspects. The execution of the algorithm in a mobile host MH can be analyzed in three distinct parts.

1. Choose own number.
2. Wait for mobile hosts with lower numbers to avail their turns.
3. Execute the critical section.

In the first part of the execution, an MH fetches the numbers chosen by other $N_{MH} - 1$ mobile hosts in order to set its own number. So, MH would

sends a query for the local numbers from other mobile hosts, and receives these numbers. As the end hosts are mobile, fetching each number involves a search for locations for both end hosts for the query as well as for delivery of the result. So, the message cost incurred for fetching a local number of another mobile host ($2C_w+C_s$). The overall the message cost incurred by MH for choosing its own number is, therefore, equal to $(N_{MH}-1)\times(2C_w+C_s)$.

In the second part of the execution, MH waits for all mobile hosts to choose their numbers and subsequently, allow those having smaller numbers to avail their turns for executing the critical section code.

Waiting part consists of two steps. It requires communication with other mobile hosts to let them choose of their resepctive numbers and subsequently allow all having a smaller number to execute critical section. At the worst, an MH has to wait till all other mobile hosts have finished choosing their respective numbers, and also all of them choose a number smaller than the MH. So, combined message complexity of waiting parts the worst can be:

$$2(N_{MH}-1)\times(2C_w+C_s).$$

The execution of third part of the code is just a local execution of instructions, and does not involve any communication with other mobile hosts. Adding the message cost of three parts, the overall communication cost for execution of Bakery algorithm only on mobile hosts is $3(N_{MH}-1)\times(2C_w+C_s)$. Therefore, a straightforward way of mapping Lamport's Bakery algorithm to mobile peer to peer distributed system leads to a message cost of the order $6N_{MH}\times C_w$.

The correctness of the algorithm requires that messages be delivered in FIFO order. This places a burden on the underlying network to maintain a logical FIFO channel between every pair of mobile hosts.

5.5.2 With Exception Machines

A system similar to the previous category, where most of the machines execute the same code except one machine (or only a few of them) that execute a different code. The machines which execute a different code are known as *exception* machines. An example of this category is Dijkstra's self stabilizing algorithm [**?**]. It is a system consisting of a set of n finite state machines connected in the form of a ring, with a *token* or previlege circulate around the ring. The possession of the token enables a machine to change its state.

Typically, for each machine, the previlege state is defined if the value of a predicate is true. The predicate is a boolean function of a machine's own state and the states of its neighbors.

The change of current state of a machine is viewed a *move*. The system is defined to be *self-stabilizing* if and only if regardless of the initial state and token selected each time, at least one token (previlege) is present and the system converges to a legal configuration after a finite number of steps. On the presence of multiple tokens in the system, the choice of which machine is entitled to make the move can be decided arbitrarily. A legal state of the system has the following properties:

- No deadlock: there must be at least one token in the system.
- Closure: every move from a legal state must place the system into a legal state. It means once system enter a legal state no future state can be illegal.
- No starvation: during an infinite execution, each machine should possess a token for an infinite number of times
- Reachability: given any two legal states, there is a series of moves that change one legal state to another.

Let us now look at Dijkstra's algorithm involving K states where $K > n$, and system consists of $n + 1$ machines. Machine 0 is called the bottom machine, and machine $n + 1$ is called the top machine. Machine are arranged in ring, where machine i has machine $i + 1 \mod (n + 1)$ as its right hand neighbor, $i = 0, 1, \ldots, n$. The legitimate states are those in which exactly one previlege is present.

For any machine, let the symbols S, L, R respectively denote the machine own state, the state of left neighbor, and the state of the right neighbor. The rules for change of states for this system are as follows.

- **Bottom machine**

  if $L = S$ then $S = (S + 1) \mod K$.

- **Other machines**

  if $L \neq S$ then $S = L$.

An initial configuration C_0 may consists of at most $n+1$ different states. At least $K-(n+1)$ states do not occur in C_0. Machine 0 increments its state only after $n+1$ steps. Therefore, it reaches a state not in initial configuration after at most $n+1$ steps. All other machines $i \neq 0$, copy states. Hence, first time machine 0 computes its state, such a state becomes unique in the ring. Machine 0 does not get a chance to compute its state until the configuration reaches $S_1 = S_2, \ldots, S_n = S_0$.

The details of how the self-stabilization works is not important for structuring distributed algorithms to mobile environment. Let us look at the communication that occurs between two machines. In self-stabilization algorithms, the essential communication perspective is to access the registers for the left and right neighbors. Thus, clearly these class of algorithms are very similar to the previous class of algorithms and the presence of one or more exception machines does not make much of a difference to the communication costs involved. However, only the neighbor's values are needed for change of state. As most of neighboring the hosts are expected to be under the same BS except for two at the edges, the overhead of wireless communication is expected to be low.

Mapping Bakery algorithm directly on a mobile distributed environment without any structuring led an energy oblivious algorithm unsuitable for mobile distributed environment. For processing just a single mutual exclusion request, one mobile host incurs a communication cost of $3N_{MH}$ messages over wireless channel. The remaining mobiles in the system collectively incur a communication cost of $3N_{MH}$ wireless messages.

Wireless interface of a mobile requires substantial power to operate. Therefore, for design of energy aware algorithms, efforts should be made to reduce the number of messsages over wireless channels. Distributed algorithms wherein each host is allowed to access local variables from every other host are expected to become energy oblivious. So, such algorithm should be structured for energy aware execution in a mobile distributed environment.

In Dijkstra's exception machine model,a single token or previlege circulates around the logical ring. In such a ring organization of mobile hosts, the communication is restricted between a host and its left or right neighbors. If we structure distributed mutual exclusion algorithm using a logical ring of cohorts, then it should be possible to reduce communication cost. The algorithm becomes very simple, we just let previleged mobile to uses access mutual exclusion. If the privileged machine is not interested, it just passes the previlege to the successor in the logical ring. This way every mobile

host gets one chance to use a critical resource in mutually exclusive manner during one full circulation of the token around the ring of mobile hosts.

The analysis of message cost of the token ring algorithm outlined above (referred to as TR-MH) is provided below.

> Both the sender and the recipient are mobile hosts. The message is sent first from the sender to its local base station then from there to the base station under which the recipient is found. So the cost of messages exchanged on wireless links is $2C_w$.
>
> The cost of locating a mobile host and subsequently sending a message to its current base station is C_s.

Therefore the cost of passing of token between two adjacent MHs in the logical ring is $2C_w + C_s$. Assuming the ring to consist of N_{MH} mobile hosts, overall cost of circulating token on the logical ring is $N_{MH}(2C_w + C_s)$. This cost is independent of the mutual exclusion requests satisfied by one complete circulation of the token in the logical ring. Let K be the number of mutual exclusion requests satisified during one complete circulation of token around the ring. The maximum number of mutual exclusion that can be met in one circulation of token is $\max\{K\} = N_{MH}$.

Each exchange of a message requires power both at the recipient and at the sender. Every MH accesses wireless twice (once for acquiring and once for releasing) during one circulation of the token through it. So, energy requirement for executing this algorithm is proportional to $2N_{MH}C_w$.

5.5.3 Coordiator Based Systems

Many distributed algorithms involve a coordinator. The coordinator bears substantially higher communication overhead compared to other participating nodes. It is basically responsible for coordination resolving issues related to synchronization. A coordinator may or may not be fixed. In a fixed coordinator based system, one node is assigned the role of the coordiator for the entire duration of execution of the algorithm. However, in a moving coordinator based system the role of coordinator can be performed by different hosts at different times. Thus the coordinator is a function of time.

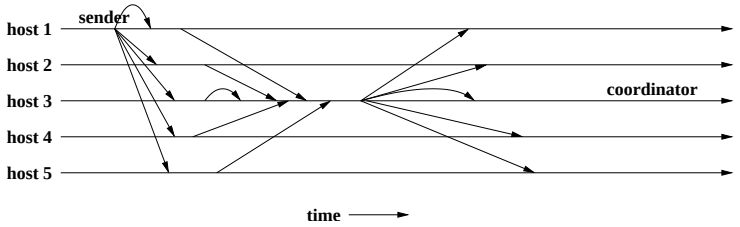

Figure 5.4: Atomic Broadcast using fixed/moving coordinator.

Fixed Coordinator Based System

Thus apart from the normal optimization for the mobile hosts, the communication pattern involving the coordinator has to be specifically optimized. Such a strategy will yield better dividends in terms of reducing the communication cost, because most of the communication load in a coordinator based system is centered around the coordinator. Apart from increased communication load, it increases the probability of failure as the coordinator is a single point of failure.

An example of the fixed coordinator system is encountered in the case of total order atomic broadcast algorithms. The system consists of N hosts, each wishes to broadcast messages to the other hosts. After a broadcast message is received a host would time stamp the message and sends it to the sequencer. On receiving the relayed broadcast message from all the nodes, the sequencer sets the time stamp of the message to the maximum of the received time stamps. This is then broadcast back to the hosts. This way a total ordering on the broadcast messages can be assured by the coordinator. Figure 5.4 illustrates this with a timeline diagram. Host 3 in figure 5.4 is the sequencer or the coordinator. The execution of the above atomic broadcast algorithm is mainly dependent on the coordinator. Therefore, in order to structure the algorithms for execution in a mobile environment, we must first turn our attention to the role of coordinator in a mobile environment. Obviously, if the coordinator is mobile then execution of any coordinated action will be expensive. So, if possible, the the coordinator should be executed on a static host, or a base station. Since, the other hosts are mobile, search, inform or proxy strategies can be applied depending on the mobility characteristics of entities in the system.

In the case where the algorithm is directly executed on the mobile hosts

without any change, the total cost incurred for each broadcast will be,

1. The cost of initial broadcast: $(N_{MH} - 1) \times (C_s + C_w)$,
2. The cost of unicasting received message from the participting nodes to the coordinator: $(N_{MH} - 1) \times (C_s + C_w)$,
3. The cost of sending time stamped messages back to participants: $(N_{MH} - 1) \times (C_s + C_w)$

Therefore, the overall messaging cost is

$$3(N_{MH} - 1) \times (C_s + C_w).$$

This can be improved marginally if the location of the coordinator cached by each base station. The cost in that case would be

$$(2(N_{MH} - 1) \times (C_s + C_w) + (N_{MH} - 1) \times (C_f + C_w)$$

However, if the coordinator is placed on a BS, the cost is revised as follows:

1. The cost of broadcast: $C_w + (N_{BS} - 1) \times C_f$,
2. The cost of unicast messages to the coordinator: $(N_{BS} - 1) \times C_f$,
3. The cost of broadcasting the timestamped messages: $(N_{BS} - 1) \times C_f + (N_{MH} - 1) \times C_w$

This leads to a overall message cost of

$$N_{MH} \times C_w + 3(N_{MH} - 1) \times C_f$$

Thus simple structuring of the atomic broadcast algorithm done by placing the coordinator on a base station leads to substantial savings in the cost of messaging.

Moving Coordinator Based System

As shown in figure 5.5 the coordinator of the algorithm changes over time. The sender, the coordinator and the receiver sets are the identical, but shown separately for the sake of clarity.

In this case normal algorithm execution at mobile hosts again has the same complexity as in the previous case. However, we can modify the system as follows:

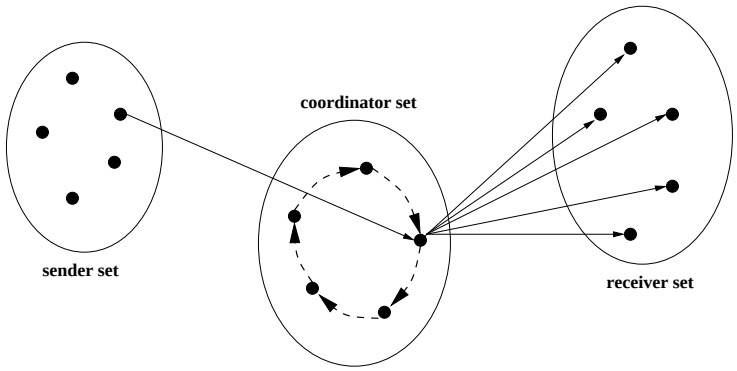

Figure 5.5: Conceptual model of a moving coordinator system

1. One of the three startegies search, inform and proxy can be used for the all the hosts in the system.

2. As soon as a mobile host becomes a coordinator, the communication load on it rises drastically in a short space of time. Hence the MH should inform its BS about change of status to coordinator, which is then broadcast to all base stations. Also the MH during its tenure as the coordinator uses the inform strategy, while other hosts use the search strategy.

Using these modifications, each step of the algorithm now requires

1. Cost of broadcast: $C_w + (N_{BS} - 1) \times C_f$,

2. Cost of sending to coordinator: $(N_{BS} - 1) \times C_f + C_w$,

3. Cost of broadcasting back the time stamped message: $(N_{BS} - 1) \times C_f + (N_{MH} - 1) \times C_w$, and

4. Additional overhead associated with change of coordinator: $C_w + (N_{BS} - 1) \times C_f$.

Thus the total cost works out as:

$$(N_{MH} + 1) \times C_w + 3(N_{BS} - 1)C_f + \alpha(C_w + (N_{BS} - 1) \times C_f),$$

where a change of the coordinator occurs every α broadcasts. The cost of executing the non-structured version of algorithm is: $3(N_{MH} - 1) \times (C_s + C_w)$. So, the saving in cost is significant by simple structuring.

5.6 Exploiting Assymmetry of Two-Tier Model

Most distributed alogrithms can be structured suitably for execution on mobile environment by reducing communication costs. Still, if we consider the token ring algorithm described in section 5.5.1, two of the key issues are not addressed, viz.,

1. Every mobile in logical ring has to maintain active network connectivity during the execution of the algorithm.

2. Relocation of computation to balance inherent asymmetry in mobile computing environment is not possible.

For example, none of the mobile hosts in the token ring algorithm can not operate either in disconnected or in doze mode during the execution. This is because the token can not be sent to a disconnected successor node, and also if the mobile holding the token decides to operate in disconnected then other mobiles may have to wait indefinitely to get their turns. The organization of distributed mobile system point towards a two-tier model with inherent asymmetry in node capabilities. The fixed nodes do not suffer from any of the resource related problem which the mobile nodes have. So, in order to balance the inherent asymmetry in the system, if the token circulation is carried by fixed hosts, then it will be possible to allow mobiles to operate in disconnected or doze mode. Furthermore, it will also be possible to relocated all compute intensive task at the fixed hosts. This strategy will not only remove the burden on resources of mobile nodes but also enhance the performance of algorithms.

Using the above two-tier approach, Badrinath, Acharya and Imielinski [1] proposed three different variations for structuring of token ring algorithm. Their main strategy was based on exploiting the inherent asymmetry in computation model as indicated design principle 1 of section 5.3.1. The token circulates over this ring in a predefined sequence. A mobile host MH wishing to access the token submits the request to its current BS. When the token becomes available, it is sent to MH at its current base station, BS′. After using the token MH returns it to BS′ which in turn returns the same back to BS. The cost of servicing token will depend on the way location of a MH is maintained.

5.6.1 Search Strategy

Pure search method scans the entire area under coverage to find a MH. The algorithm can be best understood by fragmenting the actions performed by the component devices, namely, base station and mobile hosts. Each base station maintains two separate queues, request queue Q_{req} and Q_{grant}. The token access requests by mobile hosts at a base station are queued up in Q_{req}. When the token is received by a BS from its predecessor in the logical ring, all pending requests are moved into a different queue Q_{grant}. Then all the requests are serviced from Q_{grant}, while new requests are get added to Q_{req}. After all requests are serviced, the token is passed on to the successor base station in the ring. So, the actions of BS are as follows:

```
on receipt of a request for access of the token from MH
    add MH's request to the rear of Q_req;
on receipt of the token from the predecessor in the ring {
    move all pending requests from Q_grant;
    repeat {
        remove request at head of Q_grant;
        if (MH which made the request is local to BS)
            deliver the token to MH over wireless link;
        else
            search and deliver token to MH at its current cell;
        await return of token from the MH;
    } until (Q_grant == empty);
    forward token to BS's successor in the logical ring;
}
```

As far as a mobile MH is concerned, it can request for token servicing at any time to its base station. Once token is received from the base station, it uses the critical resource and returns the token after doing so. So, the actions performed by a MH are as follows:

```
on requirement for an access of the token
    submit request to current local BS;
on receipt of token from local BS {
    hold the token and use the critical resource;
    return the token to local BS;
}
```

The correctness of algorithm relies on several implicit assumptions. Firstly, all message channels are assumed to be reliable and FIFO, so the messages actually arrive and in the order they are sent. So, a mobile sends only one request at time. Secondly, a mobile can not make a fresh request at a base station where its previous request is pending. Thirdly, the algorithm does not handle the case of missing or captured token. Fourthly, a mobile can hold token only for short finite duration of time.

With the above assumptions, it is clear that at any time only one MH holds the token. Therefore, mutual exclusion is trivially guaranteed. The token is returned to the same base station from which it is received. So, after the token is returned back a base station can continue to use it until all pending requests before arrival of token have been satisfied. The maximum number of requests that can be serviced at any base station is bounded by the size of the Q_{grant} at that base station when the token arrives. No fresh request can go to Q_{grant}, as they are added only to Q_{req}. So, a token acquired by a base station will be returned after a finite time by servicing at most all the requests which were received before the arrival of the token. This number can not exceed N_{MH}, the total number of mobile hosts in the system. It implies that the token will eventually reach each base station in the ring and, therefore, all the requests will eventually be satisfied in finite time.

It possible, however, for a mobile to get serviced multiple number time during one circulation of token over the ring. It can happen in the following way. A mobile submits a request at one base station BS and after being serviced by the token, moves quickly to the successor of BS in the logical ring and submits a fresh request for the token at the successor. Though it does not lead to starving, a stationary or slow moving mobile may have to wait for a long time to get its turn. We will look for a solution to this problem later in the text.

The communication cost of the suggested algorithm can be analyzed as follows:

1. Cost for one complete traversal of the logical ring is equal to $N_{BS} \times C_f$, where N_{BS} is the number of base stations.

2. Cost for submission of a request from a MH to a base station is C_w.

3. If the requesting MH is local to a BS receiving the token then the cost of servicing a request is C_w. But if the requesting MH has migrated

to different base station BS′ before the token reaches BS where the request was initially made, then a location search will be required. So the worstcase cost of delivering token to the requesting MH will be $C_s + C_w$.

4. The worstcase cost of returning the token to BS delivering the token to MH is $C_w + C_f$. C_f component in cost comes from the fact that MH may have subsequently migrated from BS where it made request to the cell under a different base station BS′.

Adding all the cost components, the worstcase cost of submitting a single request and satisfying is equal to $3C_w + C_f + C_s$. If K requests are met in a single traversal of the ring then the cost will be

$$K \times (3C_w + C_s + C_f) + N_{BS} \times C_f$$

Since, the number of mobile host requesting a service is much less than total number of mobiles in the system. So $K \ll N_{MH}$.

In order to evaluate the benefit of relocating computation to fixed network we have to compare it with token ring algorithm TR-MH described earlier in this section where the token circulates among mobile hosts.

- *Power consumption.* The power consumption in present algorithm is proportional to $3K$ as only $3K$ messages are exchanged over wireless links. In TR-MH algorithm it was $2N_{MH}$. As $K \ll N_{MH}$ it is expected that $\frac{3K}{2N_{MH}} < 1$
- *Search cost.* For the present algorithm it is $K \times C_s$ whereas in the previous algorithm the cost is $N_{MH} \times C_s$ which is considerably more.

5.6.2 Inform Strategy

Inform strategy reduces the search cost. It is based on simple idea that the search becomes faster if enough footprints of the search object is available. In other words, search is less expensive if more information is available about the possible location of the mobile host being searched for. Essentially, the cost of search is proportional to the amount updated location information. For example, the search cost will be 0 if a MH to notifies the BS (where it submitted its request) after every change in its location till it has received the token. So, the difference between purely search based algorithm to inform

based algorithm is that Q_{req} maintains the location area information along with request made by a mobile host, and every mobile host with a pending request informs change in location to the base station where the request is made. The algorithm specifying the actions of a BS is provide below.

on receipt of a request from a local MH:
    add the request $< MH, BS >$ to the rear of Q_{req};
on receipt of a *inform*(MH, BS') message
    replace $< MH, BS >$ in Q_{req} by $< MH, BS' >$;
on receipt of the token from the predecessor of BS in logical ring {
    move Q_{req} entries to the Q_{grant};
    repeat
        remove the request $< MH, BS' >$ at the head of Q_{grant};
        if $(BS' == BS)$
            deliver the token to MH over the local wireless link;
        else
            forward token to BS' for delivery to MH;
        await return of the token from MH;
    until (Q_{grant} == empty);
    forward token to BS's successor in logical ring;
}

The actions to be executed by a mobile host MH are provided below.

on requirement of MH needing an access to the token:
    submit request to its current local BS
    stores current local BS in the local variable *req_locn*.
on receipt of the token from BS *req_locn*:
    access the critical resource or region
    return token to BS (*req_locn*)
    set *req_locn* to $\perp$.
on each move made by MH:
    sends *join*(MH, *req_locn*) message to local BS
on entering the cell under a BS:
    if $(req\_locn \neq \perp)$
        sends a *inform*(MH, BS') to BS (*req_locn*).

To compare the search and inform strategies, let a MH submit a request at BS and receive the token at BS. Assume that it makes MOB number of

moves in the intervening period between submission of the request and the receipt of the token. After each of these moves, a *inform*() message was sent to BS, i.e. the cost of inform is $MOB \times C_f$. Since the location of MH is known after each move it makes, there is no need to search for its location. So, when token becomes available at BS and it is MH's turn to use the token, then it is directly despatched to BS$'$ where MH currently found. On the other hand, in the algorithm employing simple search, BS must search for the current location of MH, incurring a cost C_s. Therefore, the inform strategy is cost effective compared to search strategy provided $MOB \times C_f < C_s$. In other words if MH is less mobile after submitting its request, then it is better for MH to inform BS about every change of location rather than BS searching for it.

5.6.3 Proxy Strategy

Proxy strategy combines advantages of both search and inform strategies. It is designed to exploit the imbalance between frequencies of local versus global moves that a mobile host makes. Normally, a mobile host moves more frequently between adjacent cells and rarely between non-adjacent cells. Therefore, the coverage area consisting of all base stations are partitioned logically into *regions*. BSs within the same region are associated with a common proxy. A proxy is a static host, not necessarily a base station. The token circulates in a logical ring comprising of proxies. Each proxy, when it receives the token, is responsible for servicing pending requests from the request queue maintained by it. So, the proxy strategy is just a minor variation of inform strategy. Only, implicit assumption here is that a mobile host is aware of the identity of the proxy associated with its current cell. This can be realised by having each BS include the identity of its associated proxy in the periodic beacon messages. The actions executed by a proxy P are provided by the following algorithm.

on receipt of a token request from MH
    // request forwarded by a BS within P's local area),
    add request $< MH, P >$ to the rear of Q_{req};
on receipt of a *inform*(MH, P') message:
    replace request (MH, P) in Q_{req} by (MH, P');
on receipt of the token from the predecessor in logical ring {
    requests from Q_{req} moved to Q_{grant};

```
    repeat
        delete request < MH, P') > from head of Q_req;
        if (P' == P) // MH located within P's area,
            deliver the token to MH after local search;
        else // MH is in a different proxy area
            forward the token to P' which delivers it to MH;
        await return of the token from MH;
    until (Q_grant == empty)
    forward token to P's successor in ring;
}
```

The actions executed by mobile host MH are:

```
on requirement for an access of the token {
    submit request < MH, P) > to local BS;
    stores identity of local proxy in init_proxy;
}
on the receipt of the token from init_proxy {
    use the token;
    return token to init_proxy;
    set init_proxy to ⊥;
}
on a inter regional move by MH {
    send a join(MH, init_proxy) message to new BS;
    if (init_proxy ∉ {P, ⊥}) // P' is the proxy of new BS
        new BS sends a inform(MH, P') to init_proxy;
}
```

Let N_{proxy} represent the number of proxies constituting the ring. Let N_{BS} be the number of BSs in the coverage area. Then there are N_{BS}/N_{proxy} base station in a region. Let MOB_{wide} be the number of inter regional moves made by a MH in the period between submitting a token request and receiving the token. Similarly, let MOB_{local} represent the total number of intra regional moves in the same period. So, the total number of moves MOB is equal to $MOB_{wide} + MOB_{local}$. Before delivering the token to a MH, a proxy needs to locate a MH amongst the BSs within its region. This as search is referred to as local search with an associated cost C_{ls}. With the above notations and assumptions, the communication costs can be analyzed as follows:

1. Cost of one token circulation in the ring: $N_{proxy} \times C_f$

2. Cost of submitting a token request from a MH to its proxy: $C_w + C_f$
3. Cost of delivering the token to the MH: $C_f + C_{ls} + C_w$

   Note that C_f term can be dropped from above cost expression if MH receives the token in the same region where it submitted its request.
4. Cost of returning the token from the MH to the proxy: $C_w + C_f$

The above costs add up to: $3C_w + 3C_f + C_{ls}$

If an inter regional move is made, then the current local BS of MH sends the identity of new proxy to the proxy where MH submitted the request initially. The cost for inform is, therefore, C_f. The worst overall cost for satisfying a request from a MH, including the inform cost, is then

$$(3C_w + 3C_f + C_{ls}) + (MOB_{wide} \times C_f)$$

Obviously, the cost of circulating the token amongst the proxies is less than circulating it amongst all BSs by a factor N_{proxy}/N_{BS}. The cost of circulating the token within the logical ring is a measure of distribution of the overall computational workload amongst static hosts. If all three schemes service the same number of token requests in one traversal of the ring, then this workload is shared by

- N_{BS} static hosts in search strategy,
- It is shared amongst N_{proxy} fixed hosts under the proxy method.

To compare the efficiency of each algorithm to handle mobility, we need to consider the communication cost of satisfying token requests. The three algorithms incur the following costs to satisfy a request for the token:

search: $3C_w + C_f + C_s$

inform: $3C_w + C_f + (MOB \times C_f)$

proxy: $3C_w + (3 + MOB_{wide}) \times C_f + C_{ls}$

The above cost expressions can be compared against one another to determine which strategy performs better than the other. For instance, proxy strategy performs better than search strategy, if

$$\begin{array}{lcl} (3 + MOB_{wide}) \times C_f + C_{ls} & < & C_f + C_s \\ \equiv \quad MOB_{wide} + 2 & < & (C_s - C_{ls})/C_f \end{array} \tag{5.1}$$

Search strategy requires a BS to query all other BSs within a search region to determine if a MH is local to its cell. The BS currently local to the MH will respond, and the original BS can then forward the token to this BS. If N_{region} is the number of BSs within the search area, the search cost is equal to $(N_{region} + 1) \times C_f$. Using such a search strategy, where the search cost is linearly proportional to the number of locations within the search area, formula 5.1 above reduces to:

$$MOB_{wide} < N_{BS} - (N_{BS}/N_{proxy}) - 2$$

Indicating that the proxy scheme performs better than search when the number of inter regional moves made by a migrant MH is two less than the total number of BSs that is outside any given region.

Now let us compare proxy with inform. The proxy scheme incurs a lower cost to satisfy a token request when:

$$\begin{aligned} &(3 + MOB_{wide}) \times C_f + C_{ls} < (MOB + 1) \times C_f \\ &\equiv C_{ls} < (MOB - MOB_{wide} - 2) \times C_f \\ &\equiv C_{ls} < (MOB_{local} - 2) \times C_f \end{aligned} \tag{5.2}$$

The cost of a local search equals $(N_{BS}/N_{proxy}+1) \times C_f$, since all base stations have to be queried by the proxy in its region and only the local base station of MH will reply. So, the formula 5.2 above reduces to:

$$N_{BS}/N_{proxy} + 2 < MOB_{local}$$

This indicates that if the number of intra regional moves made by a migrant MH is two more than the number of BSs under each proxy, then proxy scheme outperforms the inform scheme.

Fairness in Access of the Token

There are two entities whose locations vary with time namely, the token and the mobile hosts. So we may have a situation in which a mobile host MH

1. submits a request to its current local base station BS.
2. gets the token from BS and uses it,
3. moves to the base station BS' which is the next recipient of the token,

4. submits a request for the token access at BS'.

So, the mobile host MH by virtue of being more mobile than other mobile hosts can gain multiple accesses to the token during one circulation of the token through the fixed hosts. This violates the *fairness* property of the token access among mobile hosts. So, we need to put additional synchronization mechanisms in place to make the algorithms fair. Interestingly, the problem of fairness does not arise in the algorithm TR-MH, which maintains the logical ring amongst MHs. Therefore, we need to evolve ways to preserve the functionality of fairness of TR-MH algorithm in the token algorithms which maintain logical ring within the fixed network regardless of the mobility of hosts.

Of course, none of the algorithms, search, inform and proxy cause starvation. Because a stationary mobile host is guaranteed to get its request granted when the token arrives its local base station. We have already observed that the length the pending requests in request queue at a fixed host is always bounded. This means after a finite delay each fixed host will release the token. Therefore, each requesting mobile host at a base station will eventually gain an access to the token. In the worst case, a stationary MH may gain access to the token after every other mobile host has accessed the token once from every base station, i.e, after $(N_{MH} - 1) \times N_{BS}$ requests have been satisfied. A simple fix to the problem of fairness is as follows:

1. The token is associated with a loop counter (`token_val`) which is incremented every time it completes one traversal.

2. Each MH maintains a local counter `access_count` whose current value is sent along with the MH's request for the token to the local BS.

3. A pending request is moved from the request queue to the grant queue at the BS (or the proxy) holding the token, only if the request's `access_count` is less than the token's current `token_val`.

4. When a MH receives the token, it copies the current value of `token_val` into its `access_count`.

Since a MH's `access_count` is reset to the token's current `token_val` after each access of the token, a MH can access the token only once in a single circular traversal of the token even if a MH has a low `access_count`. With the modified algorithm, the number of token accesses K satisfied in

one traversal of the ring is limited to N_{MH} (when the ring comprises of all BSs), while the value of K could be at most $N_{BS} \times N_{MH}$ otherwise. So the difference between the above modification and the original scheme essentially represents a trade-off between "fairness" of token access among the contending MHs and satisfying as many token requests as possible per circular traversal of the token.

Of course one can devise an alternative definition of "fairness" that requires a MH to be allowed to access the token multiple times in one traversal of the ring, under the limitation that its total number of accesses to the token does not exceed the current `token_val`. This criterion can be implemented by modifying step (4) above, as follows: the `access_count` of a MH is incremented on every access, instead of being assigned the current value of `token_val`.

5.7 Termination Detection

There is clear a evidence that the generic design principle of exploiting asymmetry in two-tier model would be very effective in structuring distributed algorithms for mobile environment. However, the scope of discussion in the previous section was restricted. It is just centered around design of mutual exclusion algorithm. Therefore, we need to examine further how the design principle built around the asymmetry approach could be useful for other problems as well. In this section we focus on termination detection.

Termination of a distributed computation represents one of global state of the computation. The recording global states of a distributed systems is known to be difficult. Therefore, it is difficult to record the termination state of a distributed computation. However, termination being one of the stable properties a distributed computation can be observed. When a computation terminates there can be no messages in transit. Therefore, it is possible to design an observing algorithm which does not interfere with the main computation but is able to detect termination of the computation. More precisely, the termination detection algorithm determines whether a distributed computation has entered a state of *silence*. In a silent state no process is active and all the communication channels are empty, taking into account unpredictable delays in message delivery.

An implicit assumption made by most termination detection algorithms is that the main computation never enters an incorrect state. In the case of

mobile distributed systems, termination detection is more complex, because the detection algorithm should also handle the scenarios arising out of many mobile hosts operating in disconnected mode. Mobile hosts in disconnected mode should not be disturbed. Of course voluntary disconnection can be planned so termination algorithm can handle them. But, in the cases of involuntary disconnections, mobile hosts may not regain connectivity due to failure. In other words, an incorrect state at times is indistinguishable from a correct state of the computation in mobile distributed systems.

5.7.1 Two Known Approaches

In a termination state the channels are empty. Therefore, no message can reach any of the process and consequently, no processes can become active ever again under this state. There are two fundamentally different approaches to detect termination:

- Diffusion.
- Weight throwing.

A convenient model to visualize a distributed computation is a directed graph that grows and shrinks as the computation progresses [2]. If such a graph contains an edge from a node $n1$ to another node $n2$, then $n2$ is known as a successor of $n1$, and $n1$ a predecessor of $n1$. Every node in a distributed computation starts with a a *neutral state*. Dijkstra and Scholten [2] define that a diffusion based distributed computation is initiated in a neutral state when the environment on its own generates a message and sends it to its successor. After the first message has been sent, an internal node is free to send messages to its successor. So, a diffusion computation grows as a directed graph. After an internal node has performed its node specific computation, it signals completion to its predecessors. In practice, though a two-way communication is possible, the flow of computation messages is only in the direction from a predecessor to a succcesor. The completion event can be viewed as an acknowledgement for some computation message received earlier by the node. So when completion event is eventually signaled back to environment, the distributed computation is assumed to have terminated.

In weight throwing scheme [3, 5, 8], environment or starting process in neutral state has a weight credit of 1 with it. Every message sent by any node is associated with a weight. The sending node partitions the weight

available with it into two parts. One of the part is attached to the message before sending it while the other part is retained by the sending node. The computation is started by the environment generating a message of its own and sending it to its successor. Thereafter the internal nodes send messages to their successors as in a diffusion computation. When the node specific computations at a node is over, it signals completion of computation by a weight reportage message to its predecessor besides setting its local weight to 0. The weight reportage message carries the total weight left at a node at the time of completion of the local node specific computation.

5.7.2 Approach for Mobile Distributed Systems

Termination detection in mobile distributed system follows a a hybrid approach [9]. It consists of running a simplified diffusion based termination detection algorithm for the mobile part and a weight throwing based algorithm for the fixed part. The base stations act as bridges between the two parts.

Let us first introduce a few notations which will be convenient to understand the above protocol. A mobile distributed computation can be described by:

1. A special process P_c called weight collector.
2. A set of base station processes denoted by P_i, for $i = 1, 2, \ldots$.
3. A set of mobile processes, P_i^m, for $i = 1, 2, \ldots$.
4. A set of messages.

The computation is started by the weight collector process. However, this does not necessarily represent a limitation of the model. A computation may also be triggered by a mobile process. In that case the weight collection will be performed by the static process representing the current base station for the initiating mobile process. In effect, the starting base station process becomes the environment node.

A mobile process can roam and execute handoff from one base station to another. When a mobile process moves the distributed computation on that process is suspended. If a mobile process moves away from its current base station and unable to find a new base station to which it can connect, then the mobile process is said to be temporarily disconnected. Further, it

is assumed that mobile process can not carry out any basic computation as long as it remains disconnected.

5.7.3 Message Types

Six different types of messages are needed for the algorithm. These are:

1. M_b: a basic message. If M_b is tagged with a weight w, then it is denoted by $M_b(w)$.
2. $M_{wr}(w)$: a reporting message with a weight w.
3. $M_{ack}(k)$: an acknowledgement for k basic messages.
4. M_{HF}: handoff message. It contains four subtypes: $M_{HF.req}$, $M_{HF.ind}$, $M_{HF.rep}$ and $M_{HF.ack}$.
5. M_{DIS}: message for temporary disconnection. It contains two subtypes: $M_{DIS.req}$, $M_{DIS.ind}$
6. M_{JOIN}: messages connected with rejoining of a disconnected mobile node. It consist of two subtypes: $M_{JOIN.ind}$ and $M_{JOIN.rep}$

Termination detection in a mobile distributed system is of two types:

(i) Strong termination, and

(ii) Weak termination.

A strong termination state is reached when all processes have turned idle, there are no disconnected mobile process or any basic message in transit. In a weak termination state all processes except disconnected mobile processes have reached the state as in strong termination.

The protocol for termination detection in mobile systems should allow for a disconnected process to rejoin the computation at a later point of time. It is possible that a mobile may be disconnected due to failure. In that case the mobile may not join back. If the termination detection protocol does not have a provision to handle such a situation, and then it will not work. Weak termination is an important indication of anticipated system failure due to disconnected mobile processes. So, roll back and recovery protocols can be planned around conditions involving weak termination.

The termination detection algorithm proposed by Tseng and Tan [9] divides the protocol into several parts and specifies it in form of the actions by a mobile process, a base station process and the weight collector process. Before we discuss these algorithms it is necessary to understand how the mobile processes communicate with static processes and while mobile hosts roams.

A mobile process always receives a message from a base station. But it can send a message either to its local base station or to another mobile process. The message, however, is always routed through the base station to which the mobile is connected.

When a mobile moves to another base station while being active then it requires a handoff. A handoff is initiated by sending a $M_{HF.req}$ message. The responsibility for carrying out the rest of the handoff process lies with the static part. Apart of sending handoff, a disconnected mobile node may send a rejoin message on finding a suitable base station to connect with. The rejoin message, denoted by $M_{JOIN.req}$, is sent to the base station with which mobile node attempts to connect. The only other message that a mobile node could send is a signal for completion of local computation. This message is sent to base station. After sending this the mobile turns idle. So the relevant message types handled by a mobile host (MH) are:

- M_b: a basic message which one MH sends to another. M_b is always routed through the base station of the sender MH. A base station after receiving a message M_b from a mobile host (MH), appends a weight w to M_b and sends $M_b(w)$ to the base station of the receiving MH.
- $M_{HF.req}$: This message is sent by an MH to its local base station for a handoff request.
- $M_{JOIN.req}$: This message is sent by a disconnected MH when it is able to hear radio beacons from a base station of the network.

A base station can send a basic message either to a mobile node or to another base station on behalf of a connected mobile. The base stations are also involved in handoffs to facilitate roaming of mobiles. A disconnected mobile may rejoin when it hears beacons from a base station over its radio interface. In order to perform its tasks, a base station has to process and send different message types of messages. These message and intended use of these messages can be understood in the context of the protocol discussed later.

5.7.4 Entities and Overview of Their Actions

The termination detection protocol is carried out by actions of following three entities:

- Mobile nodes,
- Base stations, and
- Weight collector.

The termination detection protocol requires support at protocol level for roaming and disconnected mobile processes. A roaming of mobile process is managed by handoff protocol. Handoff protocol handles the transfer of weights associated with basic computation messages to appropriate base station. Similarly, the concerned base stations transfers associated weights of disconnected mobile processes to the weight collector. The transfer of weights as indicated above ensures that every base station can correctly transfer the weights to the weight collector when completion of basic computation is signalled.

The protocol distinguishes between two sets of mobile processes:

- MP_i: set of active mobile processes under base station BS_i.
- DP: set of disconnected mobile processes in the system.

Each active mobile processes is associated with an active mobile node while each disconnected process is associated with one disconnected mobile nodes. The set of mobile processes involved in the computation is given by the union $DP \cup_i \{MP_i\}$.

5.7.5 Mobile Process

A mobile process (which runs on a mobile host) can either receive a basic message from a base station or send a basic message to another mobile process. All messages either originating or terminating at a mobile host (MH) are routed through the current base station of MH. When a mobile process receives a basic message it keeps a count of the number of unacknowledged message received from the concerned base station. So, mobile process knows the number of acknowledgements to be sent to the base station when it has finished computation.

A mobile process P_j^m always routes a basic message M_b through its current base station process P_i to another mobile process P_k^m. The process P_i attaches a weight $w_i/2$ to M_b and sends $M_b(w_i/2)$ to the base station hosting P_k^m. P_i also keeps track of the acknowledgements it has received from all mobile processes. So no additional protocol level support is required at P_j^m for sending a basic message to any other process P_k^m.

P_j^m increments the number of unacknowledged messages by 1 when it receives a basic message from P_i. A local counter in is used by a mobile process to update the count of unacknowledged messages. For example, P_j^m knows that it has to send acknowledgements for in messages. Therefore, before turning idle, P_j^m signals the completion of basic computation by sending an acknowledgement for in messages to P_i. After the acknowledgement has been sent, in is set to 0. The summary of the rules applicable for a mobile node in the termination detection protocol are as follows:

Rule M1: For sending a basic message to another process no additional operation needed.

Rule M2: On receiving a basic message from base station:

$in = in + 1;$

Rule M3: On turning idle from active mode, execute the following:

send $M_{ack}(in)$ to current base station;
$in = 0;$

5.7.6 Base Stations

Every message to and from a mobile process P_j^m is routed always through P_i, its current base station process. P_i attaches a weight x from its available weights to every message M_b from P_j^m before sending to the base station process P_l for the destination. P_l receives $M_b(x) = M_b + \{x\}$. The process P_l extracts the weight x from $M_b(x)$ and adds x to its the current weight. After that message $M_b = M_b(x) - \{x\}$ is forwarded to P_k^m. This way any base station process can control the weight throwing part of the protocol on behalf of all the mobile processes interacts with.

In order to control the diffusion part, a mobile process P_i keeps track of the messages it has sent to every mobile process P_j^m under it. P_i expects a mobile process P_j^m under it to send acknowledgement for each and every

message P_j^m has received. P_j^m signals the completion of computation by sending the acknowledgements for all the messages of P_i. So, when all P_j^ms involved in a computation have sent their final acknowledgement to P_i, the termination in diffusion part is detected.

The rules for termination detection executed by a base station process P_i are as follows:

Rule B1: P_i on receiving a basic message $M_b(x)$ from wired network which is meant for P_j^m does the following:

$w_i = w_i + x$; // Extract weight carried by $M_b(x)$
$M_b = M_b(x) - \{x\}$;
$out_i[j] = out_i[j] + 1$;
forward M_b (without appending x) to P_j^m;

Rule B2: On receiving a basic message M_b from P_j^m on wireless network destined for a process P_k^m, it does the following:

if ($P_k^m \in MP_i$) { // P_k^m is local mobile process under P_i
    $out_i[k] = out_i[k] + 1$;
    forward M_b to P_k^m;
}**else** { // Destination P_k^m is non local
    $M_b(w_i/2) = M_b + w_i/2$; //Attach weight to M_b
    $w_i^b = w_i^b/2$;
    locate MH_k's base station BS_{ell}; // Base station process P_{ell}
    send $M_b(w_i/2)$ to P_ℓ;
}

Rule B3: On receiving $M_{ack}(k)$ from $P_j^m \in MP_i$, P_i executes:

$out_i[j] = out_i[j] - k$;
if ($out_i[k] == 0$ for all $P_k^m \in MP_i$) {
    sends $M_{wr}(w_i)$ to P_c;
    $w_i = 0$;
}

Before we consider handoff and rejoin protocols, let us examine two important properties of the protocol specified by actions of three entities we have described so far.

Property 1 *If $out_i[j] = 0$ then P_j^m is idle and there is no in-transit basic messages between P_i and P_j^m.*

Proof: When $out_i[j] = 0$, all basic messages sent by P_i to P_j^m have been acknowledged. So P_j^m must be idle. As every channel is assumed to be FIFO, if message M is sent before M' over a channel, then M must be received before M' at the other end. When P_j^m turns idle it sends acknowledgement for all messages it has received from P_i. So when this acknowledgement (which the last among the sequence of messages between P_i and P_j^m) is received by P_i there can be no message in-transit message on the channel because the acknowledgement would flush all other messages before it.

Property 2 *If $w_i = 0$ then all mobile processes $P_j^m \in MP_i$ are idle and there is no in-transit basic message within P_i*

Proof: If $w_i = 0$, then the weights held by P_i on behalf of all mobile process $P_j^m \in MP_i$ have been sent back to P_c. This can happen only if $out_i[j] = 0$, for all $P_j^m \in MP_i$. By Property 1, it implies P_j^m is idle and there is no in-transit basic message between P_i and P_j^m. Hence the property holds.

5.7.7 Handoff

When a mobile process P_j^m moves from its current cell under a base station BS_i to a new cell under another base station BS_k then handoff should be executed. As a part of the handoff process, the relevant data structure and the procedure for managing termination detection protocol concerning P_j^m should be passed on to BS_k.

P_j^m makes a request for handoff to BS_k and waits for the acknowledgement from BS_k. If the acknowledgement is received then P_j^m starts interacting with BS_k for initiating transfer of the data structure related P_j^m from BS_i to BS_k. If handoff acknowledgement is not received within a timeout period, P_j^m retries handoff. The handoff procedure is a slight variation from layer-2 handoff. In a cellular based wireless network, when a handoff is initiated, old base station BS_{old} sends handoff request to mobile switching center (MSC). MSC then forwards the request to new base station BS_{new} (determined through signal measurements). BS_{old} then accepts the request and handoff command is executed through BS_{old}. So, for protocol level support from handoff, the mobile process should send $M_{HF.req}$ to BS_i identifying BS_k. BS_i accepts the

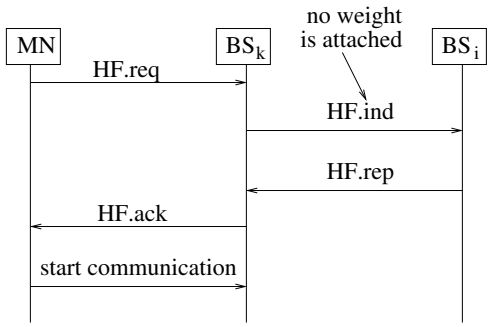

Figure 5.6: Illustration of handoff process

request and also sends $M_{HF.ack}$ to P_j^m. After this P_j^m starts communicating with BS_k. BS_i, before sending $M_{HF.ack}$ to P_j^m, also transfers the needed data structure to BS_k. The negotiation for transfer of link between BS_i and BS_k may involve messages like $M_{HF.ind}$ and $M_{HF.rep}$ between BS_i and BS_k. The handoff process has been illustrated in Figure 5.6. To keep the protocol simple, P_j^m has been shown to directly approach BS_k for executing handoff. In response to handoff indication message from BS_k, process at BS_i sends a message to BS_k that includes:

- Count of the number of unacknowledged messages in respect of mobile process P_j^m. These are the basic messages for which BS_i was expecting acknowledgements at the time P_j^m sought a handoff.
- A weight $w = w_i/2$.

The base station process P_i at BS_i should check if P_k^m is the last mobile process under it. In that case P_i should itself turn idle. But before turning idle, it should send its existing weight credit to weight collector process.

Once $M(w_i/2)$ has been received at BS_k, it should update its own data structure for keeping track of mobile processes. Thus the summary of the protocol rules for handoff execution are as indicated below.

Rule H1: On finding handoff condition P_j^m does the following:

sends a $M_{HF.req}$ message to P_k and waits for $M_{HF.ack}$;
if ($M.HF.ack$ received from P_k)
starts communicating with P_k; // With base station BS_k

else // After time out
retries handoff;

Rule H2: On receiving $M_{HF.req}$ from P_j^m, P_k:

sends a $M_{HF.ind}(P_j^m)$ to P_i; // No weight attached.

Rule H3: On receiving $M_{HF.ind}(P_j^m)$ from P_k, P_i executes:

sends a $M_{HF.rep}(P_j^m, out_i[j], w_i/2)$ to P_k;
$MP_i = MP_i - \{P_j^m\}$; // P_j^m no longer under P_i
$w_i = w_i/2$;
if ($out_i[l] = 0$ for all $l \in MP_i$) { // P_j^m is the last mobile process under P_i
sends $M_{wr}(w_i)$ to P_c;
$w_i = 0$;
}

Rule H4: On receiving $M_{HF.rep}(P_j^m, out_i[j], w_i/2)$ from P_i, P_k does the following:

$MP_k = MP_k \cup \{P_j^m\}$;
$out_k[j] = out_i[j]$;
$w_k = w_k + w_i/2$;
sends a $M_{HF.ack}$ to P_j^m;

5.7.8 Disconnection and Rejoining

It is assumed that all disconnections are planned. However, this is not a limitation of the protocol. Unplanned disconnection can be detected by timeouts and hello beacons. When a mobile process P_j^j is planning a disconnection it sends $M_{DISC.req}$ message to its current base station BS_i. Then BS_i suspends all communication with P_j^m and sends $M_{DISC.ind}(P_j^m, out_i[j], w_i/2)$ to P_c. It will allow P_c to detect a weak termination condition if one arises.

The rules for handling disconnection are, thus, as follows:

Rule D1: Before P_j^m enters a disconnected mode it

sends $M_{DISC.req}$ message to P_i;
suspends all basic computation;

Rule D2: On receiving $M_{DISC.req}$ from P_j^m, P_i executes:

suspends all communication with P_j^m;
sends a $M_{DISC.ind}(P_j^m, out_i[j], w_i/2)$ to P_c;
$MP_i = MP_i - \{P_j^m\}$;
$w_i = w_i/2$;
if ($out_i[k] == 0$ for all $P_k^m \in MP_i$) {
    sends $M_{wr}(w_i)$ to P_c;
    $w_i = 0$;
}

Rule D3: On receiving $M_{DISC.ind}(P_j^m, out_i[j], w_i/2)$ from P_i, P_c executes:

$DP = DP - \{P_j^m\}$;
$out_c[j] = out_i[j]$;
$w_{DIS} = w_{DIS} + w_i/2$;

Final part of the protocol is for handling rejoining. If a mobile process can hear the radio of signals from a base station it can rejoin that base station. For performing a rejoin P_j^m has to execute the instructions given below.

Rule R1: A disconnected mobile process P_j^m on entering a new cell under BS_i does the following

sends $M_{JOIN.req}$ message to P_i and waits;
if ($M_{JOIN.ack}$ is received from P_i)
    starts basic computation;
else // After timeout
retries JOIN;

Rule R2: On receiving $M_{JOIN.req}$ from P_j^m, P_i executes:

sends a $M_{JOIN.ind}(P_j^m)$ to P_c; // No weight attached

Rule R3: On receiving $M_{JOIN.ind}(P_j^m)$ from P_i, P_c executes:

if ($P_j^m \in DP$) {
    sends a $M_{JOIN.rep}(P_j^m, out_c[j], w_{DIS}/2)$ to P_i;
    $DP = DP - \{P_j^m\}$;
    if ($DP = \Phi$);
        $w_c = w_c + w_{DIS}$;
}

Rule R4: On receiving $M_{JOIN.rep}(P_j^m, out_c[j], w_{DIS}/2)$ from P_i, P_c does the following:

$MP_i^b = MP_i^b + \{P_j^m\}$;
$out_i[j] = out_c[j]$;
sends a $M_{JOIN.ack}$ to P_j^m;
restarts communication with P_j^m;

5.7.9 Dangling Messages

There may be messages in system which can not be delivered due to disconnection or handoff. Such dangling messages should be delivered if the mobile process reconnects at a later point of time. So, care must be taken to handle these messages. When a mobile process is involved in a handoff it can not deliver any message to static host or base station. So, the mobile process hold such undelivered messages with it until handoff is complete. This way message exchange history is correctly recorded. Dangling messages at base stations are those destined for mobile processes involved in handoff or disconnection. These messages are associated with weights which is ensured by weight throwing part of the algorithm. So, handling dangling messages becomes easy. All dangling messages from base station processes are sent to weight collector. The weight collector process holds the messages for future processing.

Rule D1: P_c on receiving a dangling message carrying a weight x:
$w_{DIS} = w_{DIS} + x$;

Rule D2: On P_c sending a dangling message M_b to reconnected P_j^m:
appends $w_{DIS}/2$ to M_b;
$w_{DIS} = w_{DIS}/2$;
send $M_b(w_{DIS}/2)$ to P_i;// $P_j^m \in MP_i$
if (dangling message list == Φ) {// No disconnected mobiles exist
$w_c = w_c + w_{DIS}$; // Reclaim residual weight
$w_{DIS} = 0$;
}

5.7.10 Announcing Termination

The termination is detected by P_c when it finds $w_c + w_{DIS} = 1$. If $w_c = 1$ then it is a case of strong termination, otherwise it is case of weak termination. In the case of weak termination $w_{DIS} > 0$.

Bibliography

[1] B.R.Badrinath, Acharya, A., and Imielinski, T. Designing distributed algorithms for mobile computing networks. *Computer Communications 19*, 4 (1996), 309–320.

[2] Dijkstra, E. W., and Scholten, C. S. Termination detection for diffusing computations. *Information Processing Letters 11* (1980), 1–4.

[3] Huang, S. T. Detecting termination of distributed computations by external agents. In *International Conference on Distributed Computing Systems* (1989), pp. 79–84.

[4] LeLann, G. Distributed systems towards a formal approach. In *IFIP Congress on Information Processing* (1977), vol. 7, pp. 155–160.

[5] Mattern, F. Golbal quiescence detection based on credit distribution and recovery. *Information Processing Letters 30* (1989), 195–200.

[6] Murphy, A. L., Roman, G.-C., and Varghese, G. An algorithm for message delivery to mobile units. In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC'97)* (ACM Press, 1997).

[7] R. G. Gallager, P. H., and Spira, P. A distributed algorithm for minimum weight spanning trees. *ACM Transaction on Programming Languages and Systems 5* (January 1983).

[8] Tseng, Y.-C. Detecting termination by weight-throwing in a faulty distributed system. *Journal of Parallel and Distributed Computing 25* (1995), 7–15.

[9] Tseng, Y.-C., and Tan, C.-C. Termination detection protocols for mobile distributed systems. *IEEE Transaction on Parallel and Distributed Systems 12*, 6 (2001), 558–566.