Recovery in Distributed Mobile Environments*

P. Krishna N. H. Vaidya D. K. Pradhan

Computer Science Department
Texas A&M University
College Station, TX 77843-3112

Abstract

Mobile computing is a rapidly emerging trend in distributed computing. The new mobile computing environment presents many challenges due to the mobile nature of the hosts. In this paper we present some fault-tolerant data management strategies for a distributed mobile environment. These strategies need to be different from the traditional fault-tolerance approaches because of the resource limitations of mobile computing environment.

1 Introduction

Mobile computers using wireless networks is a rapidly emerging trend in computer systems. This will give users the information accessing capability regardless of the location of the user or of the information. Nowadays, the portable computers are as powerful as some desktop workstations in terms of computing power, memory, display and disk storage [7]. Users of portable computers would like to carry their laptops with them whenever they move from one place to another and yet maintain transparent network access through the wireless link. With the availability of wireless interface cards, the mobile users are no longer required to remain confined within the static network premises to get network access.

In this paper we present some recovery strategies for mobile systems. Traditional fault-tolerance schemes cannot be directly applied to these systems. The mobile computing environment poses challenging fault-tolerant data management problems due to the mobility of the users, limited bandwidth on the wireless link, and power restrictions on the mobile hosts. Mobile systems are more often subject to environmental conditions which can cause loss of communications or data. Because of the consumer orientation of most mobile systems, run-time faults must be corrected with minimal (if any) intervention from the user. Therefore, the fault-tolerance capability must be self-contained.

2 System Model

In this section we present a model for a distributed system with mobile hosts. As shown in Figure 1, mobile networks are generally composed of a static backbone network and a wireless network. There are two distinct sets of entities, namely, mobile hosts and fixed hosts. A host that can move while retaining its network connection is a mobile host [6]. The static network comprises of the fixed hosts and the communication links between them. Some of the fixed hosts, called *base stations* (*BS*) are augmented with a wireless interface, and, they provide a gateway for communication between the wireless network and the static network. There are different types of wireless networks, namely, cellular (analog and digital cellular phones), wireless LAN, and unused portions of FM radio or satellite services (paging) [6]. The same mobile host can interact with all three different types of wireless networks at different points of time. Due to the limited range of the wireless transreceivers, a mobile host can communicate with a *base station* only within a limited geographical region around it. This region is referred to as a base station's *cell*. The geographical area covered by a *cell* is a function of the medium used for wireless communication. Currently, the average size of a cell is of the order of 1-2 miles in diameter [6]. In future, the cells are expected to be nano-cellular (less than 10 meters in diameter) covering the interior of a building [7]. A mobile host can reside in the *cell* of only one *base station* at any given time. Due to mobility, the mobile host may cross the

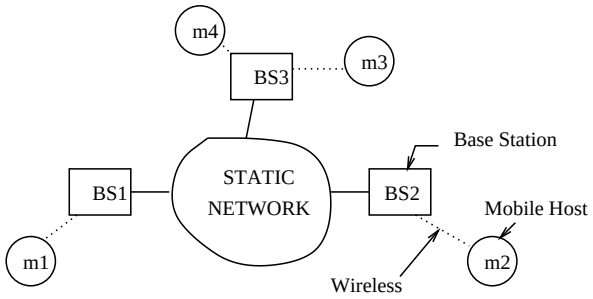

Figure 1: Model of a distributed system with mobile hosts

boundary between two cells while being active. Thus, the task of forwarding data between the static network and the mobile host must be transferred to the new cell's *base station*. This process, known as *handoff* is

*Research reported is supported in part by AFOSR under grant F49620-92-J-0383.

transparent to the mobile user [7]. It, thus, maintains end-to-end connectivity in the dynamically reconfigured network topology.

3 Modes of Failure

In this paper we will be mainly dealing with the recovery of the mobile computing system upon the failure of a mobile unit[1]. Mobile systems are more often subject to environmental adversities which can cause loss of communications or data. There are several modes of mobile unit "failure", namely, 1) failure of the mobile host, 2) disconnection of the mobile host, and 3) message loss due to weak wireless link. In this paper we develop strategies to tolerate failure of the mobile host, i.e. mode 1. This is because it is this mode that affects the system state, whereas failure modes 2 and 3 mainly delay the system response.

- Failure of the mobile host : In this mode the mobile host fails resulting in a loss of its volatile state. The mobile host is assumed to be *fail-stop* i.e., the *base station* is able to detect the failure of the mobile host. One way to detect the failure, is to make the active mobile host send a "I am alive" message at regular intervals of time to the *base station*.

- Disconnection of the mobile host : Mobile units run on batteries. Smaller units run on AA batteries; larger units run on Ni-Cd packs [6]. Due to battery power restrictions, the mobile units will be frequently disconnected (powered off). The difference between disconnection and failure is its *elective* nature. Disconnections can be treated as *planned* failures which can be anticipated and prepared for [6].

- Weak wireless link : The *base station* and the mobile hosts are connected by a slow and often unreliable wireless link. This is equivalent to an intermittently faulty link, which transmits the correct message during fault-free conditions, and stops transmitting upon a failure.

4 Motivation

A distributed system with mobile units can be considered to have a two tier structure: reliable static network with fixed hosts and a number of mobile units connected by a slow and often unreliable wireless link. The new *style* brought about by mobile computing, poses new challenges to the *fault-tolerant* data management community.

Consider Figure 2 as an example, where base stations *BS1* and *BS2* are a part of a static network. *BS1* and *BS2* cover the cells which contain mobile hosts *m1* and *m2* respectively. Hosts *m1* and *m2* may be taking part in a distributed algorithm, or a critical transaction, or may be querying each other. If the mobile hosts were taking part in a critical task, the task might have to be aborted due to the absence of fault-tolerance. For instance, let *m2* perform critical data processing, and it requires some updated data from *m1* before it can proceed. If *m1* fails, *m2* will have to abort the data processing. A mechanism for recovering an intermediate *consistent* state of the system may be useful to avoid loosing all the work performed by a task.

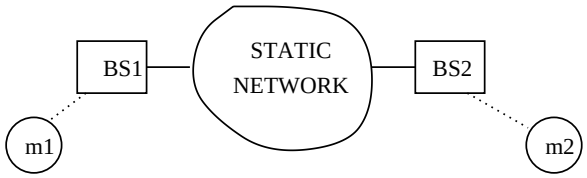

Figure 2: Distributed processing between mobile hosts

Over the past couple of years there has been a lot of literature dealing with locating strategies, and modifying distributed algorithms for "reliable" mobile hosts [6, 13, 2, 3]. But how do we recover the execution (with or without real-time deadlines) without user intervention, when one of the participants (i.e. mobile host) fails? Is the cost of incorporating fault-tolerance, during normal fault-free execution, high? What are the tradeoffs? We are trying to answer these questions, and this paper presents some preliminary ideas.

Traditional *fault-tolerance* schemes like checkpointing and message logging [8, 4, 11, 12, 9] would have served the purpose, without any modification, if the mobile hosts restricted their movements within one *cell*. But, that is not the case - because the whole concept of mobile computing revolves around the fact that the computing environment does not require a user to maintain a fixed position in the network and it allows almost unrestricted user mobility. Due to the mobility, the mobile users cross cells, and *handoff* occurs. In addition to the mobility of the user, we must keep in mind the battery power restrictions of the mobile hosts, and also the limited bandwidth of the wireless connection. The bandwidth limitations restrict the volume of data that can be transferred over the wireless link. Keeping in mind these new resource limitations, we have tried to develop strategies for "energy efficient" fault-tolerant data management in a distributed mobile computing environment.

5 Base Station Driven Recovery Strategies

In the following subsections we are going to present two strategies for saving the state, and two strategies for *handoff*. The strategies for saving the state are similar to the traditional fault-tolerance strategies.

5.1 State Saving

We present two strategies to back-up the process state: (i) *Synchronous* and (ii) *Asynchronous*. In both strategies, basically, the *base station* of the cell in which the mobile host currently resides, serves as a *back-up* for that host. For recovery purposes, it is not a good idea to store the log of transactions or messages on the mobile host, as mobile host state is lost on its failure. Hence we will require that the process image and the log be backed up on the *base station*.

[1] In the following section we will briefly explain the recovery of the system upon the failure of a *base station*.

As stated earlier, the mobile hosts might be taking part in some distributed application. Such applications require messages to be transferred between the mobile hosts, and might also require user inputs. While the user inputs may go directly to the mobile host, the messages will first reach the *base station* in charge of the *cell* in which the mobile host currently resides. The *base station*, then forwards it to the corresponding mobile host. Likewise, all the messages sent by a mobile host, will be first sent to its *base station*, which will forward it over the static network to the *base station* which is in charge of the *cell* in which the destination mobile host currently resides. Figure 3 presents the message types and transfers that may take place in a distributed mobile environment.

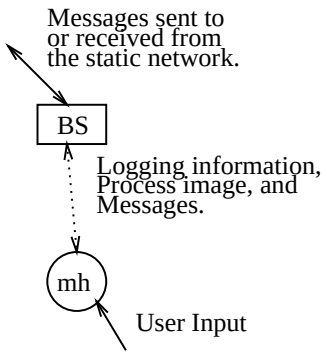

Figure 3: Message transfer between the fixed and the mobile host

The process image is backed up periodically at a stable storage - in our case it is the *base station*, since we assume the *base station* to be fault-tolerant. Saving the state periodically has been one of the traditional fault-tolerance measures, to avoid recomputing/reexecuting from the initial state. Upon a failure, the process rolls back to the most recent *back-up* process image and continues execution; thus resulting in loss of execution only from the most recent *back-up* process image. Systems with hard real-time constraints require a recovery mechanism that minimizes the loss of computation. Thus, the question "how frequently do we *back-up* the process image on the *base station*?" arises. We present two strategies for backing up the process image - depending upon the application requirements, the user might want to choose one of them. The two strategies are described below. (Note: Instead of backing up the whole process image, we might just want to save the "incremental" process image - thereby saving some bandwidth.) Both the strategies allow each mobile host to fail and recover independently. Also, the processes are assumed to be deterministic.

5.1.1 Synchronous Approach

The state of the process can get altered either upon receipt of a message from another mobile host or upon an user input. The messages or inputs that modify the state are called *write*[2] events. In the *synchronous* approach, the "incremental" process image of the mobile host is backed up at the *base station* upon every *write* event on the mobile host data.

Upon a failure of the mobile host, the *base station* loads the latest backed up process image. Thus, the recovery will be quick and totally transparent to the external environment (which comprises of other hosts taking part in the distributed algorithm or transaction etc.). Upon repair, the mobile host sends a message to the *base station*, which then loads the latest process image onto the mobile host, and allows the mobile host to continue from that point.

As stated earlier, *handoff* occurs only when an *active* mobile host crosses cells. Failure after *handoff* can be recovered using the strategies described in the next section. But there can be a situation where a host fails, then goes to another cell, and then recovers. In this case, *handoff* does not happen, thus, the new cell's *base station* does not know the last cell of the mobile host. So recovery is not possible. This situation can be avoided if the *base station* of the cell in which the mobile host fails notifies the *home location server* [6] of the mobile host. Thus when the mobile host recovers in a new cell, its *base station*, if it does not know the last cell, asks the *home location server* of the mobile host. Thus upon knowing the last cell, the *base station* sends a message requesting the process image of the mobile host from the last cell's *base station*. Some variations of this scheme also exist.

Synchronous approach is most suited for systems with mobile hosts having a very high failure rate. In addition to this, this approach will be suited for systems having hard real-time deadlines, thus requiring a quick recovery upon failure. But, the current state of art for wireless communications supports limited bandwidth, thereby restricting the volume of data that can be transferred over the wireless medium [6]. Therefore, instead of backing up the process image synchronously (i.e. after every *write* event), we might want to do the *back-up* asynchronously – as explained in the next section.

5.1.2 Asynchronous Approach

In this strategy we *back-up* the process image after regular intervals. The intervals can be determined either by certain number of (a) *write* events, or (b) time units. As defined earlier, the messages or inputs that modify the state of the mobile host are called *write* events. The length of the interval is to be determined from the application requirements and the failure rate of the mobile host. If the failure rate is high, we might want to *back-up* more frequently. Likewise, if we have real-time constraints we must ensure a quick recovery - thus *back-up* more frequently.

Backing up the process image is not enough for recovery. In addition, we will have to log the messages or transaction (depending on the application type). If

[2] If semantics of the message is not known, in the worst case, we might have to assume that the process image gets altered upon receipt of every message or an user input.

a write message is received from another mobile host, the *base station* first logs it and then forwards it to the mobile host for execution. Likewise, upon an user input (write event), the mobile host first forwards a copy of the user input to the base station for logging. After logging, the *base station* sends an acknowledgement back to the mobile host. The mobile host can process the input, while waiting for the acknowledgement, but not send a response to the user. Only upon receipt of the acknowledgement, the mobile host sends its response.

The above procedure ensures that no messages or user inputs are lost due to a failure of the mobile host. The logging of the write events continue till a new process image is backed up at the *base station*. The *base station* then purges the log of the old write events.

Upon failure of the mobile host, the *base station* loads the latest backed up process image of the mobile host and restarts executing by replaying the write requests from its logs, thus reaching the state of the mobile host before failure.

5.2 Handoff

Till now, the recovery strategy seems to be very similar to the traditional fault-tolerance schemes. But, for mobile systems, in addition to above we have to deal with *handoff* process. What should happen when the mobile user moves to a new cell in the middle of a distributed algorithm's execution or a transaction's execution?

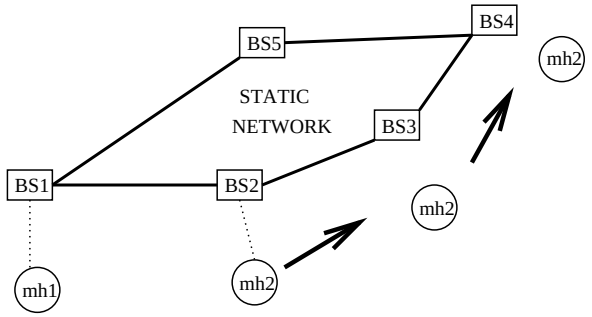

Figure 4: Handoff in the middle of an execution

Consider Figure 4 as an example. Here, mobile hosts *mh1* and *mh2* are executing a distributed algorithm. In the middle of the execution, *mh2* moves through the cell of *BS3* and settles in the cell of *BS4*. Handoff occurs at the boundaries of *BS2* and *BS3*, and, *BS3* and *BS4*. The mobile host *mh2* fails on reaching the cell of *BS4*. If *mh2* had remained in the cell of *BS2* the system would have recovered because the process image and the logs are backed up at *BS2*. But since no *back-up* took place at *BS3* or *BS4*, and since *BS4* does not know where the last *back-up* of *mh2* is stored, recovery cannot take place. Thus, the execution has to be aborted. We tackle this problem by basically transferring some information corresponding to the mobile host during the handoff process. There are two ways in which data can be transferred during the *handoff* process, (i) *Pessimistic* and (ii) *Lazy*.

5.2.1 Pessimistic Strategy

The process image and log of write requests are transferred immediately to the new cell's *base station*. Upon receipt of the process image and the log, the new cell's *base station* sends an acknowledgement to the old *base station*. The old *base station*, on receiving the acknowledgement, deletes its copy of the process image and the log, since the mobile host is no longer in its cell. This is done to save space on the *base station*. Consider Figure 5 as an example.

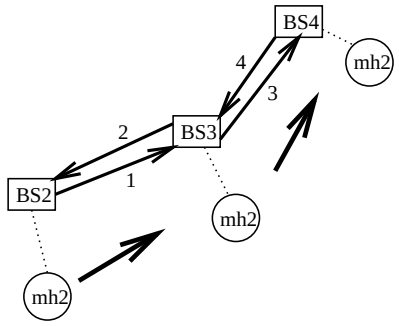

Figure 5: Pessimistic handoff

When *mh2* crosses the boundary of *BS2* and *BS3*, handoff occurs. *BS2* sends the process image and logs corresponding to *mh2* (1). *BS3*, on receiving the data, sends back an acknowledgement to *BS2* (2). *BS2*, then, deletes the process image and the logs of *mh2*. Likewise, when *mh2* crosses the boundary of *BS3* and *BS4*, at the end of steps (3,4), the process image and logs corresponding to *mh2* is present at *BS4*. Thus, if *mh2* fails in this cell, successful and quick recovery is attained.

This strategy will be suited for distributed systems where the failure rate of the mobile hosts are very high, and for applications where long service disruptions are not tolerated. The disadvantage with this approach is that there will be heavy volume of data transfer during each handoff. This can be avoided if we use the *Lazy* strategy, as explained in the next section.

5.2.2 Lazy Strategy

With *Lazy* strategy, during handoff, there is no transfer of process image and the log. Consider Figure 6 as an example. Upon a handoff of *mh2* from *BS2* to *BS3*, *BS2* sends a message to *BS3* indicating the last cell location of *mh2* i.e. the cell of *BS2* (1). Similarly, when *mh2* moves into the cell of *BS4*, *BS3* sends an indication to *BS4* (2). The new cell's *base station* just remembers the mobile host's last cell. Thus, as a mobile host moves from cell to cell, the corresponding *base stations* effectively form a linked list. One such linked list needs to be maintained for each mobile host.

This strategy might lead to a problem, because, the process image and logs of *mh2* may be unnecessarily saved at different base stations. Thus to avoid this, after a handoff, if a *back-up* is done at the new base station, it sends a notification to the last cell's base

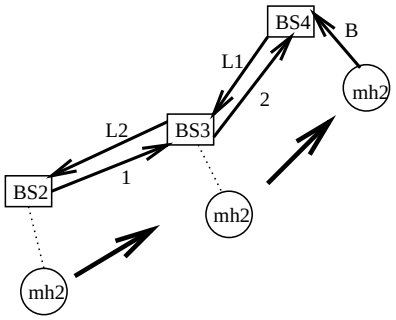

Figure 6: Lazy handoff

station to delete the process image and logs of *mh2*, if present. If not present, this base station forwards the notification to predecessor's base station in the linked list. This process continues, till a base station with a old process image and logs of *mh2* is encountered or the starting cell of *mh2* is reached. In Figure 6, after a *back-up* operation at *BS4* (B), *BS4* sends a notification to *BS3* (L1), which in turn forwards the notification to *BS2* (L2) (because there was no *back-up* or logging done at *BS3*). Incidentally, there was a *back-up* done at *BS2*. Thus, *BS2* deletes the process image and log corresponding to *mh2*.

The *Lazy* strategy saves a lot of time during handoff as compared to the *Pessimistic* strategy. But the recovery is more complicated. Upon a failure, if the *base station* does not have a *back-up* of the process image, it gets the logs and the process image from the *base stations* in the linked list. The base station then loads the process image and replays the messages from the logs to reach the state of the mobile host just before failure. The recovery is presented in the form of a pseudo-code in Figure 7.

Recovery()

If Current BS has the latest process image
  Load process image and replay messages from the log to recover.
else
  Request the BS's in the linked list for the process image and log.
  Load process image and replay messages from the logs to recover.

Figure 7: Recovery when using *Lazy()* handoff

5.3 Tradeoff Parameters

Depending on the system specifications and requirements, the appropriate recovery and *handoff* strategy will be chosen. The tradeoff will depend on the following factors, (i) failure rate of the mobile host, (ii) communication/mobility ratio, (iii) message size (energy efficiency), (iv) memory constraints on the *base station* and (v) time available for recovery.

- Failure rate of the host : System failures are caused by defects introduced in manufacturing or by transient or permanent faults occurring during operation. Moreover the mobile systems are subject to environmental conditions which can cause loss of communications or data. For systems with a high failure rate, a *Synchronous* state saving strategy may be preferred, if a quick recovery is required.

- Communication/Mobility ratio : Communication refers to the number of messages sent/received by the mobile host and mobility refers to the number of moves the mobile host makes in a given period of time. For mobile hosts having low communication/mobility ratio (a very mobile user), the *Lazy* handoff strategy may be preferred.

- Message Size : The constraint of limited available energy will require "energy efficient" data management strategies. Transmitting and receiving data consumes additional power. In general, transmitting a given amount of data consumes twice as much power as receiving the same amount of data [6]. Thus, systems having severe energy constraints, *Asynchronous* state saving strategy may be preferred.

- Memory constraints : A *base station* generally has many mobile hosts in its cell. Storing the process image and the log of each mobile host at the *base station* might use up a lot of memory space on the *base station*. It is necessary to evaluate average memory requirements based on statistical data and the recovery schemes used. Generally, the process image, logs, and the linked list for each mobile host will be stored in the *base station* memory.

- Recovery time : This is essentially the time required to recover a process upon failure. If the process has hard real-time deadlines, or requires high availability, the recovery upon a failure should be quick. For quick recovery, a combination of *Synchronous* state saving and *Pessimistic* handoff strategy may be preferred.

We are currently studying the strategies using these tradeoff parameters to find out the environments where a particular combination of recovery and handoff strategy will be best suited.

5.4 Failure of the Base Station

Till now, we have been assuming the *base station* to be fault-free. How do we recover the system if a *base station* fails? We assume the *base station* to be *fail-stop* [10]. Since the *base station* is a part of the static network, traditional fault-tolerance schemes like checkpointing and logging can be used with minor modifications to support the communication with the mobile hosts. A simple solution to this problem is "replication". Recovery due to the failure of a *base station* can be achieved by switching in a back-up *base station* (similar to the "cold-spare" approach of Tandem [5]).

6 Conclusion

Mobile computing is a rapidly emerging trend in distributed computing. The new mobile computing environment presents many challenges due to the requirements of the mobile nature of the hosts. In this paper we have presented preliminary ideas for fault-tolerant data management in a distributed mobile computing environment. These strategies need to be different from the traditional fault-tolerance approaches, because of the resource limitations of mobile computing environment. We have also identified the tradeoff parameters to evaluate the recovery scheme. We are currently studying the strategies using these tradeoff parameters to determine the environments where a particular combination of recovery and handoff strategy will be best suited.

References

[1] B. R. Badrinath and T. Imielinski, "Replication and Mobility," *Proc. of Second Workshop on the Management of Replicated Data,* November 1992.

[2] B. R. Badrinath, A. Acharya and T. Imielinski, "Structuring distributed algorithms for mobile hosts," Technical Report, WINLAB/Rutgers Tech. Rept DCS-TR-298/WINLAB TR-55.

[3] P. Bhagwat and C. E. Perkins, "A mobile networking system based on internet protocol(IP)," *Proc. of the USENIX Symposium on Mobile and Location-Independent Computing,* Cambridge, Massachussets, August 1993.

[4] B. Bhargava and S. R. Lian, "Independent checkpointing and concurrent rollback recovery for recovery - An optimistic approach," *Proc. IEEE Symp. on Reliable Distributed Systems,* pp. 3-12, 1988.

[5] C. I. Dimmer, "The Tandem non-stop system," *Resilient Computing Systems,* edited by T. Anderson, chapter 10, pp. 178-196, Collins, London, 1985.

[6] T. Imielinski and B. R. Badrinath, "Mobile wireless computing: solutions and challenges in data management," Technical Report, Rutgers DCS-TR-296/WINLAB TR-49, Feb. 1993.

[7] K. Keeton et.al., "Providing connection-oriented network services to mobile hosts," *Proc. of the USENIX Symposium on Mobile and Location-Independent Computing,* Cambridge, Massachussets, August 1993.

[8] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. on Software Engineering,* Vol. SE-13, No. 1, pp. 23-31, Jan. 1987.

[9] P. Krishna, Nitin H. Vaidya and D. K. Pradhan, "Independent checkpointing and recovery scheme for fail-slow processors," Tech. Rep. 93-028, Dept. of Computer Science, Texas A&M University, 1993.

[10] R. D. Schlichting and F. B. Schneider, "Fail-stop processors : An approach to designing fault-tolerant distributed computing systems," *ACM Trans. on Computer Systems,* Vol. 1, No. 3, pp. 222-238, Aug. 1983.

[11] R. E. Strom and S. Yemini, "Optimisitic recovery in distributed systems," *ACM Trans. on Comp. Sys.,* pp. 204-226, Aug. 1985.

[12] Nitin H. Vaidya, "Dynamic cluster-based recovery: Pessimistic and Optimistic Schemes," Tech. Rep. 93-027, Dept. of Computer Science, Texas A&M University, 1993.

[13] T. Watson and B. N. Bershad, "Local area mobile computing on stock hardware and mostly stock software," *Proc. of the USENIX Symposium on Mobile and Location-Independent Computing,* Cambridge, Massachussets, August 1993.