A Domino-Effect Free Recovery Mechanism for Mobile Computing Environment

A. Vutukuru, B. Gupta, N. Mogharreban, S. Rahimi and L. Tallam
Department of Computer Science
Southern Illinois University Carbondale
Carbondale, IL-62901 USA

Abstract - In this work we have proposed a recovery scheme suitable for mobile computing environment. The salient features of this scheme are the following. After the system recovers from failures, only the faulty processes rollback and restart from their respective recent checkpoints while the fault free processes continue their normal operation; it is independent of the number of processes that may fail concurrently; it handles appropriately both orphan and lost messages and it uses simple data structures to maintain the total order of lost messages to be replayed to a faulty process after its recovery.

Keywords: Orphan Messages, Lost Messages, Mobile Computing

1 Introduction

A distributed system consists of a collection of distinct processes which are spatially separated and communicate with one another by exchanging messages [1]. A computation on a distributed system is performed by dividing the task, and by executing each part simultaneously on multiple processes in a way, such that they appear as a single system. A Mobile Computing system is a Distributed System in which at least one of the processes is mobile. Checkpointing and rollback recovery is a popular method to achieve fault-tolerance in a distributed computing system. For recovery, the system needs to roll back to a consistent global state [2]. To make sure that a distributed application results in correct computation after the system recovers from a failure, both the orphan and lost messages are to be considered along with the duplicate and delayed messages. Note that the messages for which the sender is not known but the receiver is known is termed as an orphan message and the messages for which the sender is known but the receiver is not known is termed as lost message.

The two fundamental approaches for checkpointing and recovery approaches are (i) Asynchronous Checkpointing and (ii) Synchronous Checkpointing. In Asynchronous Checkpointing, a process can take a checkpoint periodically without any coordination among the processes. After recovery from a failure, a failed processor rolls back to its recent checkpoint and communicates with the dependent processes to build a consistent state. Due to the absence of coordination among the processes, this recovery suffers from domino effect. Notable work to cope with the domino effect has been reported which use the concept of message logging [3], [6]. In Synchronous check pointing approach, processes send checkpoint coordination messages to synchronize their checkpointing activities. Due to coordination, a global consistent state is always maintained and a domino free recovery is achieved. In this approach, during check pointing coordination, process execution may be suspended which results in performance degradation and due to high message overhead, it is not suitable for mobile computing systems. It may be noted that in case of mobile computing environment, the efficient use of the limited resources, viz. limited wireless bandwidth, mobile hosts' limited battery power and memory must have to be taken into consideration while designing a check pointing and recovery algorithm [4], [5], [7]-[9].

Problem formulation: The objective of the present work is to design a mechanism for recovery in mobile computing environment which will ensure efficient use of the limited resources, viz., limited battery power, memory, and bandwidth of mobile computing environment. This recovery mechanism will handle both orphan and lost messages as well as concurrent failures. To achieve this objective we will design an effective message logging mechanism which is neither pessimistic nor optimistic. We also assume asynchronous check pointing.
This paper is organized as follows: in Sections 2 and 3 we have stated the system model and the relevant data structures respectively. In Section 4, we have explained the working principle of message logging and mobility handling mechanisms. Recovery mechanism is stated in Section 5. In Section 6, the working principles of message logging, mobility and recovery are explained with an example. In Section 7, performance of the proposed recovery mechanism is compared with the existing works. Finally, Section 8 draws the conclusion.

2 System model

The Mobile computing environment consisting of Mobile support stations (MSS) and Mobile Hosts (MH) has the following characteristics: The MSSs are reliable and are interconnected by a wired network. They communicate via messages sent through the wired channels. Reliable communication is assumed between the MSSs. Each MSS has a fixed wireless transmission range known as a cell and an MH can move from one cell to another. At any time, an MH can be connected to at most one MSS. Message passing between two hosts in different cells is enabled via the MSSs to which they are connected. We also assume that processes are deterministic and fail-stop.

Because of the limited bandwidth of the wireless channels and the limited battery power, memory of the mobile hosts as well as their unreliability, we assume that the tasks of message logging, storing the checkpoint information, and location management are handled by the reliable MSSs.

3 Relevant data structures

We use the following notations in describing our approach. The i\textsuperscript{th} Mobile Host is denoted as h\textsubscript{i} and the p\textsuperscript{th} Mobile Support Station is denoted by S\textsubscript{p}. An application message generated by h\textsubscript{i} is denoted as m\textsubscript{(i, s)} where m is the message, and s is the sequence number (send_seq) of the message sent by the host h\textsubscript{i}. The a\textsuperscript{th} checkpoint taken by h\textsubscript{i} is denoted by C\textsubscript{a}.h\textsubscript{i}.

3.1 Data structures maintained at each mobile host

(i) Send Sequence number [send_seq]: Initially set to Zero and is incremented by one when a new application message is generated by a Mobile Host.

(ii) Receive Sequence number [recv_seq]: Initially set to Zero and is incremented by one when a new application message is received by a Mobile Host.

(iii) Trace: Trace contains three integer variables [cp_seq, cp_loc, lr], where cp_seq denotes the recent checkpoint sequence number of the MH, cp_loc denotes the id of the MSS which contains the recent checkpoint and lr denotes the recv_seq number of the last application message received by the MH before the recent checkpoint.Trace of h\textsubscript{i} is denoted by Trace\textsubscript{i}. Whenever an MH takes a checkpoint, it copies the Trace values in the MSS to which it is connected at that time.

3.2 Data structures maintained at each mobile support station

(i) Active_List: Active_List contains the list of MH ids that are supported in the cells associated to different MSSs. This is maintained at all the support stations in the network.

(ii) Identifier: When an application message say m\textsubscript{(i, s)} arrives at an MSS, say S\textsubscript{p} from an MH, say h\textsubscript{i} connected to it, then S\textsubscript{p} assigns an Identifier to this message as M\textsubscript{(i, j, k)}, where 'i' is the id of the h\textsubscript{i} (sender), 'j' is the id of h\textsubscript{j} (destination) and 'k' denotes the k\textsuperscript{th} message sent by h\textsubscript{i} to h\textsubscript{j}.

(iii) Message Log [msg_log]: Message Log contains [SeqN, Sent_Order, Receive_Order, MSG];

SeqN denotes the sequence number for the inter cell and intra cell application messages arrived at an MSS. SeqN is assigned by the receiving MSS. It is useful in maintaining the total order of the messages arriving at the MSS.

Sent_Order contains [Identifier, SSN] where SSN denotes the sequence number for the messages received from a particular host (sender) in the home cell. SSN is initially set to Zero and is incremented by one when a new application message arrives from a particular host in the home cell.

Receive_Order contains [Identifier, RSN], where RSN denotes the sequence number for the messages received by a particular host (destination) in the home cell. RSN is initially set to Zero and is incremented by one when a new application message arrives at the MSS for a particular host in the home cell.
MSG is the application message arrived at the MSS. This message can be logged either with the Send Order (inter cell message for a host in another cell) or Receive Order (inter cell message to be delivered to a host residing in the home cell) or with both (intra cell message).

3.3 Types of messages

Application Messages: Application Messages are those which are passed between the MHs during computation.

Control Messages: Following are the different control messages used in this work.
1. Message Request [msg_req]: If a message is not received in order, then msg_req is sent by an MH to a MSS or vice-versa to request for a correct message.
2. Log Request [log_req]: It is used by an MSS to retrieve the message log entries for a new MH after it has joined its cell. The log entries of the mobile host with RSN greater than the latest rec_seq (lr) in its recent checkpoint are retrieved from the msg_log of the MSS to which the MH was previously connected.
3. Recovery Request [reco_req]: Upon recovery after a failure, an MH sends reco_req to the MSS to which it is connected. When the MSS receives the reco_req, it replies with the recent checkpoint information of the MH stored in the Trace and the MH rolls back to its recent checkpoint.

Messages Related to Mobility Handling:
4. Leave (ls, lr): When an MH connected to an MSS leaves its current cell; it sends Leave (ls, lr) to the MSS. It contains the latest send_seq (ls) and latest rec_seq (lr).
5. Join (ls, lr): It is sent by the MH to the new MSS so as to establish the connection and it contains the latest send_seq (ls) and latest rec_seq (lr) of the MH before it has left its previous MSS. MH attaches its id and Trace along with the Join.

4 Message logging and mobility handling

4.1 Message logging mechanism

Let us assume that h_i is connected to S_p (its home cell) and h_j is connected to S_q (its home cell). The mechanism of passing an application message from h_i to h_j is stated below.

When S_p receives an application message m_{(i, s)} from h_i

S_p checks the send_seq associated with the m_{(i, s)}
If (send_seq(s) = SSN + 1)
   // maintaining the order of the messages generated by h_i
   { Accepts the Message; Increments SSN of h_i; Sets Identifier to m_{i(k)} as M_{i(k)}; Locates the Destination MH from Active List; Logs the message with the Send_Order in msg_log by assigning SeqN; S_p attaches the Identifier to the message and passes it to S_q; // an inter cell message }
   
   Else
   { Discards the message; Sends msg_req to the host h_i to send the message having (send_seq=SSN + 1); }

When S_p receives an application message for h_i from S_q

Increments RSN of h_i; Logs the message with the Receive_Order of the msg_log by assigning SeqN; // an inter cell message S_q attaches the RSN to the message and passes it to h_i; // when h_i receives message from S_q
If (current rec_seq + 1 = RSN) Accepts the Message;
Else
   { Discards the message; Sends msg_req to MSS to send the message with (RSN = (rec_seq + 1));

4.2 Updating of Trace_i during checkpointing

When h_i connected to S_p takes a checkpoint say C_i, the Trace_i is updated in the following way.

Update the Trace_i cp_seq=a //checkpoint sequence number of host h_i cp_loc=S_p // id of MSS to which MH is connected lr // latest rec_seq of the h_i
Copy the Trace_i in S_p
4.3 Mobility handling

The stepwise summarization for mobility handling of \( h_i \) when it moves from the cell supported by \( S_p \) and joins the cell supported by \( S_q \) is stated below.

1. \( h_i \) sends Leave (ls, lr) to \( S_p \) and \( S_p \) broadcasts the Leave (ls, lr) to the other MSSs in the network

   // this enables the MSSs in the network to know that \( h_i \) has left \( S_p \). Messages heading to \( h_i \) are logged and are passed to \( h_i \) after its new location is known.

2. \( h_i \) sends Join (ls, lr) and Trace, to \( S_q \) along with its id // when \( h_i \) joins the cell of \( S_q \)

3. \( S_q \) adds \( h_i \) to its Active_List and broadcasts the Join (ls, lr) to the other MSS in the network

   // this enables the MSSs in the network to know the new location of \( h_i \)

4. \( S_q \) sends log_req to the \( S_p \) and \( S_p \) transfers the log entries of \( h_i \) with RSN \( \geq \) lr in its recent checkpoint

5. \( S_q \) copies these entries in its msg_log without assigning SeqN.

6. Messages with RSN greater than lr (in Join) are delivered to \( h_i \) in order.

7. \( S_q \) updates the SSN of \( h_i \) equal to lr in the Join and RSN equal to the RSN of the last message sent to \( h_i \)

8. Messages heading to \( h_i \) are now sent to \( S_q \) and \( S_q \) delivers them to \( h_i \) in order.

5 Recovery

The MHs are fail-stop and due to mobility, an MH can fail either in its home cell or in a foreign cell. Each faulty process (an MH) after recovery from a failure rolls back to its recent checkpoint (i.e. domino-effect free recovery) independent of what other processes (faulty or not) are doing and the MSS to which this faulty MH is connected replays the messages required for correct computation after its recovery. After receiving these messages, the MH restarts its normal computation. Thus observe that this recovery approach is independent of the number of processes that may fail concurrently. Also note that the effect of a failure is restricted only to the failed process and does not block any fault-free process's operation. The reason for this is that message loss does not occur in case of any fault-free process and also each such fault-free process can detect any duplicate message that may be sent by a process after its recovery. The recovery mechanism is stated below.

5.1 Recovery of an MH residing in its home cell

When \( h_i \) fails in the cell supported by \( S_p \) (its home cell), after its recovery the following actions are taken.

\( h_i \) sends reco_req to \( S_p \);
\( S_p \) replies with the recent checkpoint from Trace;
\( h_i \) rolls back to its recent checkpoint;

Handling of Lost Messages

\( S_p \) checks the lr in the Trace,;
Replays to \( h_i \) the messages with RSN (> lr in Trace,) in order; // using RSN lost messages are replayed to \( h_i \) in the order of their arrival at \( S_p \)

Mechanism to handle Orphan and Duplicate Messages

\( S_p \) receives a message from \( h_i \) (after \( h_i \) restarts from its recent checkpoint)
\( S_p \) compares the send_seq (s) of the message with the highest SSN of \( h_i \);

If \( s \leq \) highest SSN of \( h_i \)
Discards the message;
//"s' less than or equal to highest SSN indicates that message has been handled Prior to the failure (i.e. Duplicate Message)
Else
Accepts the message;
// message has not been handled prior to the failure

5.2 Recovery of an MH in a foreign cell

When \( h_i \) fails in the cell supported by \( S_q \) (foreign cell) after moving from \( S_p \) (home cell), the recovery is performed as follows. After recovery from the failure, \( h_i \) sends reco_req to \( S_q \) and \( S_q \) replies with \( h_i \)'s recent Trace, (sent with \( h_i \)'s Join request). \( h_i \) rolls back to its recent checkpoint and \( S_q \) replays the messages with RSN greater than the lr in \( h_i \)'s recent Trace, in order (messages with RSN greater than lr in \( h_i \)'s recent checkpoint are copied
in the msg_log of $S_0$. After receiving these messages, $h_i$ restarts. When the messages generated by $h_i$ arrive at $S_p$, $S_q$ compares the send_seq (s) with the highest SSN of $h_i$ present in its msg_log. If 's' of message is less than or equal to the highest SSN of $h_i$, then the message has been handled prior to the failure of $h_i$ and $S_q$ discards it as a duplicate message; otherwise, the message would be accepted.

6 An Illustration

We now explain the mechanism of message logging, checkpointing, mobility handling, and recovery using the following example. Let us consider a mobile environment (shown in Fig. 1) in which mobile hosts $h_i$, $h_k$ are connected to $S_p$ and $h_j$ is connected to $S_q$. The arrival order of various messages generated by all the hosts arriving at the support stations is shown in Table 1. All the messages that are generated by the mobile hosts passing through support stations will be assigned with their corresponding identifiers by their respective support stations. Table 2 shows them in detail. Tables 3 and 4 describe the message logs maintained at support stations $S_p$ and $S_q$ respectively. In Fig. 1, L denotes Leave request and J denotes Join request sent by $h_i$.

During normal computation, message logging is performed as follows. The first application message generated by $h_i$ is denoted as $m(i,1)$ where $j$ represents the id of the MH and 1 represents the send_seq (s) of the message. When $m(i,1)$ arrives at $S_p$, $S_q$ checks the condition if the send_seq associated with the messages is equal to the SSN of $h_i$ plus 1 (to maintain the correct order of messages from $h_i$). Here SSN of $h_i$ is equal to 0 as $S_q$ has not received any messages from $h_i$ prior to $m(i,1)$. If the condition is true, $S_q$ accepts the message and increments the SSN of $h_i$ by 1 and sets an Identifier to the message as $M(i,1)$ (first message being sent from $h_i$ to $h_j$). $S_q$ determines the location of the destination MH (i.e., $h_i$) from the Active List and logs it with the Sent_Order (inter cell message) in the msg_log and assigns SeqN equal to 1 (first message arrived at $S_0$).

![Fig. 1 A simple mobile computing system](image)

--- Sending an Application Message from MH to MSS
--- Delivering an Application Message by MSS to MH
--- Passing an Application Message between the MSSs
--- Control Message

<table>
<thead>
<tr>
<th>Table 1: Message arrival Order at MSS</th>
<th>Table 2: Messages and their Corresponding Identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_p$</td>
<td>$S_q$</td>
</tr>
<tr>
<td>$m(i,1)$</td>
<td>$M(i,1)$</td>
</tr>
<tr>
<td>$m(i,1)$</td>
<td>$M(i,1)$</td>
</tr>
<tr>
<td>$m(i,2)$</td>
<td>$M(i,2)$</td>
</tr>
<tr>
<td>$m(i,3)$</td>
<td>$M(i,3)$</td>
</tr>
<tr>
<td>$m(k,1)$</td>
<td>$M(k,1)$</td>
</tr>
<tr>
<td>$m(k,2)$</td>
<td>$M(k,2)$</td>
</tr>
<tr>
<td>$m(k,3)$</td>
<td>$M(k,3)$</td>
</tr>
<tr>
<td>$m(j,1)$</td>
<td>$M(j,1)$</td>
</tr>
<tr>
<td>$m(j,2)$</td>
<td>$M(j,2)$</td>
</tr>
<tr>
<td>$m(j,3)$</td>
<td>$M(j,3)$</td>
</tr>
</tbody>
</table>

After logging the message, $S_q$ attaches the Identifier to the message and forwards it to $S_p$. Upon receiving the message, $S_p$ increments the RSN of $h_i$ from 1 to 2 as a message ($m(i,1)$) has arrived at $S_p$ for $h_i$ prior to $m(i,1)$. $S_p$ logs this message in the Receive_Order by assigning SeqN (2) and forwards the message to $h_i$ by attaching the RSN (2). When the message arrives at $h_i$, $h_i$ checks if the RSN attached with the message is equal to current rec_seq (1) plus 1 (to maintain the correct order of messages from $S_q$). In this case the condition is true and $h_i$ accepts the message. In case of an intra cell message like $m(k,1)$, the message is logged with both the Sent_Order and Receive_Order. The message logging of different messages arriving at the support stations are shown in detail in Table 3 and Table 4. When $h_i$ takes a checkpoint $C_1$, the Trace is updated as Trace (cp_seq = 1, cp_loc = $S_p$, lr = 1) and is copied in $S_p$.

For mobility handling let us consider that $h_i$ now moves. When $h_i$ moves from $S_p$, it sends Leave (0,2) to $S_p$ where 0 represents the $h_i$'s latest
send_seq and 2 represents the latest rec_seq, i.e. at the time of leaving from the cell supported by \( S_p \), \( h_i \) has not generated any messages and has received two messages. \( S_p \) broadcasts Leave request to other MSSs in the network. After receiving the Leave request, message \( m_{(0, 2)} \) arrives at \( S_p \) which is logged with the Receive_Order (SeqN = 4 and RSN of \( h_i = 3 \)) in the msg_log of \( S_p \). \( h_i \) sends Join (0, 2) to \( S_q \) at the time of joining the cell supported by \( S_p \). \( S_q \) adds \( h_i \) in its Active_List and broadcasts this Join request to other MSSs in the network indicating the new location of \( h_i \). First \( S_q \) sends log_req to \( S_p \) and \( S_p \) transfers msg_log entries of \( h_i \) with RSN greater than the \( lr (= 1) \) in its recent Trace. \( S_q \) copies these entries in its msg_log without assigning SeqN. \( S_q \) first sends the messages with RSN greater than \( lr \) (in the Join request) in order, i.e. \( S_q \) first delivers the message \( m_{(0, 3)} \) to \( h_i \) and updates the SSN of \( h_i \) equal to 0 and RSN of \( h_i \) equal to 3. After joining the cell supported by \( S_p \), the new messages that are bound for \( h_i \) arrive at \( S_q \) and \( S_q \) delivers them to \( h_i \) in order.

Table 3: Message Log at \( S_p \)

<table>
<thead>
<tr>
<th>SeqN</th>
<th>Sent_Order</th>
<th>Receive_Order</th>
<th>MSG</th>
</tr>
</thead>
</table>
| 1    | \( m_{(0,1),1} \) | \( m_{(0,1),1} \) | msg>
| 2    | \( \emptyset \) | \( m_{(0,1),2} \) | msg>
| 3    | \( \emptyset \) | \( m_{(0,1),1} \) | msg>
| 4    | \( \emptyset \) | \( m_{(0,2),3} \) | msg>
| 5    | \( m_{(0,3),2} \) | \( \emptyset \) | msg>

Table 4: Message Log at \( S_q \)

<table>
<thead>
<tr>
<th>SeqN</th>
<th>Sent_Order</th>
<th>Receive_Order</th>
<th>MSG</th>
</tr>
</thead>
</table>
| 1    | \( m_{(0,1),1} \) | \( \emptyset \) | msg>
| 2    | \( m_{(0,1),2} \) | \( \emptyset \) | msg>
| 3    | \( m_{(0,2),3} \) | \( \emptyset \) | msg>
| 4    | \( \emptyset \) | \( m_{(0,1),1} \) | msg>
| 5    | \( \emptyset \) | \( m_{(0,2),3} \) | msg>

We now explain the recovery mechanism. For recovery of an MH in its home cell, we consider the failure of \( h_k \) and for recovery of an MH in the foreign cell, failure of \( h_l \) is considered. When \( h_k \) recovers from failure, it sends reco_req to \( S_p \) and \( S_p \) replies with its recent checkpoint information stored in \( \text{Trace}_k \) (cp_seq=1, cp_loc=S_p, lr=0). Based on this, \( h_k \) rolls back to its recent checkpoint (Cr). Due to the roll back, the message \( m_{(0, 2)} \) becomes an orphan message and message \( m_{(j, 2)} \) becomes a lost message. First \( S_p \) replays the message with RSN greater than \( lr \) in \( \text{Trace}_p \) in order, i.e., message \( m_{(j, 2)} \) is replayed to \( h_i \). After receiving \( m_{(0, 2)} \), \( h_i \) restarts and generates \( m_{(k, 2)} \). When \( m_{(k, 2)} \) arrives at \( S_p \), \( S_p \) compares the send_seq of this message (s=2) with the highest SSN of \( k (=2) \) to check if \( m_{(k, 2)} \) has been handled prior to failure. In this case, the condition \( s \leq \text{highest SSN} \) indicates that \( m_{(k, 2)} \) has been handled prior to failure. Hence, it will be discarded as a duplicate message.

Similarly when \( h_i \) recovers from failure, it sends reco_req to \( S_q \) and \( S_q \) replies with \( h_i \)'s recent Trace, sent with the Join request. \( h_i \) rolls back to its recent checkpoint (i.e. \( C_r \)). Due to roll back, \( m_{(j, 1)} \) becomes an orphan message and, \( m_{(0, 1)} \) and \( m_{(j, 3)} \) become lost messages. When \( h_i \) joins the \( S_q \)'s cell, \( S_q \) copies the messages with RSN greater than \( lr \) (in \( C_r \)) in its msg_log. In this case, \( S_q \) first replays the messages \( m_{(j, 1)} \) and \( m_{(j, 3)} \) to \( h_i \). After receiving these messages, \( h_i \) restarts and generates \( m_{(j, 1)} \). When \( m_{(j, 1)} \) arrives at \( S_p \), \( S_p \) compares the send_seq (s=1) with the highest SSN of \( h_i \) (=1) and discards it as a duplicate message.

7 Performance

The following are the important features of the proposed recovery approach. Our work does not require the communication over the wireless channels to be FIFO. The correct order of the messages is maintained by using simple data structures based on sequence numbers. The recovery mechanism appropriately handles both the orphan and lost messages along with the duplicate and delayed messages. It also uses simple data structures; thereby ensuring efficient memory utilization. The fact that only the faulty processes after their recovery roll back to their respective recent checkpoints while the fault free processes can continue with their normal operation, ensures very efficient use of the limited battery power of the mobile hosts. Moreover, with very small amount of piggybacked control information, it ensures the efficient use of the limited wireless bandwidth. Finally, the recovery scheme is independent of the number of processes that may fail concurrently.

It may be noted that in [7], authors have assumed reliable FIFO communication over the wireless channels. Our work is independent of any such restriction. The work in [8] offers an efficient domino-effect free recovery approach where processes restart from their respective recent
checkpoints. However, it does not consider any lost messages and mobility of a process. In our approach, we have taken care of the lost messages as well as mobility of every MH. Also note that our approach needs much less number of the control messages than in [5], [9]. Besides, the amount of data structures needed in our approach is much less than the same in the above mentioned works.

8 Conclusion

In this work we have proposed a recovery approach suitable for mobile computing environment for handling orphan and lost messages along with the duplicate and delayed messages. The main feature of the proposed recovery algorithm is that it is domino effect free and only a failed process rolls back to its recent checkpoint while a fault free process can continue its normal operation. Also it is independent of the number of processes that may fail concurrently. The use of simple data structures, small number of control messages, small amount of piggybacked control information along with the domino effect free recovery help in the efficient use of the limited resources like bandwidth, battery power, and memory of mobile computing environment.

9 References