International Journal of Information Technology and Knowledge Management
January-June 2012, Volume 5, No. 1, pp. 219-223

INTELLIGENT MOBILE ENVIRONNEMENTS RECOVERY TECHNIQUES

Apekshit Sharma[1] and Chandrakant Sharma[2]

ABSTRACT: Checkpointing is one of the commonly used techniques to provide fault tolerance in distributed systems so that the system can operate even if one or more components have failed. However, mobile computing systems are constrained by low bandwidth, mobility, lack of stable storage, frequent disconnections and limited battery life. Hence checkpointing protocols which have fewer checkpoints are preferred in mobile environment. Since MHs are prone to failure, so they have to transfer a large amount of checkpoint data and control information to its local MSS which increases bandwidth overhead. In this paper we propose a new soft checkpointing scheme for mobile computing systems. These soft checkpoints are saved locally in the host. Locally stored checkpoints do not consume network bandwidth and can be created in less time. Before disconnecting from the MSS, these soft checkpoints are converted to hard checkpoints and are sent MSSs stable storage. In this way, taking soft checkpoint avoids overhead transferring large amount of data to the stable storage MSSs.

Keywords: Mobile distributed system, coordinated checkpointing, fault tolerance, Mobile Host

1. INTRODUCTION

A mobile distributed system consists of both Mobile Hosts (MH) and static Mobile Service Stations (MSS). A set of dynamic and wireless communication links can be established between an MH and an MSS, and a set of high-speed communication link is assumed between the MSSs. An MSS may communicate with a number of MHs but an MH at a time communicates with only one MSS. An MH communicates with the rest of the system via the MSS it is connected to. Message transmission through wireless links takes an unpredictable but finite amount of time. Reliable message delivery is assumed during normal operation. The system does not have any shared memory or global clock [1]. Distributed computation in such mobile computing environment is performed by a set of processes executing concurrently on MHs and MSSs in the network. The processes communicate asynchronously with each other. A process experiences a sequence of state transitions during its execution and the atomic action which causes the state transition is called an event. The event having no interaction with another process is called an internal event; the message sending and receipt are external events. Computation is a sequence of state transitions within a process.

The diversity and flexibility introduced by mobile computing brings new challenges to the area of fault tolerance. Types of failures that were rare in the fixed environments are common with mobile hosts. Physical damage becomes much more probable, because mobile hosts are carried with the users while they move between sites. Mobile hosts can also be lost or stolen. Transient failures due to power or connectivity problems can be frequent events.

In this paper, we focus on checkpoint based recovery technique for mobile computing systems. A checkpoint protocol typically functions as follows: The protocol periodically saves the state of the application on stable storage. When a failure occurs, the application rolls back to the last saved state and then restarts its execution. Checkpoint protocols proposed in the literature are not suitable for mobile environments because of disconnections. Another problem is that these previously proposed protocols do not adapt their behaviour to the characteristics of the current network connection. If the network has a poor Quality of Service like small bandwidth and a high failure rate, the protocol should be able to trade off recovery time with operational costs.

2. PROBLEM FORMULATION

In mobile distributed system multiple MHs are connected with their local MSS through wireless links. During checkpointing, an MH has to transfer a large of amount of data like control variables, register values, environment variable to its local MSS over the wireless network. So, it consumes resources like bandwidth, energy, time and computation power.

Mobile host failures can be separated into two different categories. The first one includes all failures that cannot be repaired; for example, the mobile host falls and breaks, or is lost or stolen. The second category contains the failures that do not permanently damage the mobile host; for example, the battery is discharged and the memory contents are lost, or the operating system crashes. The first type of failure will

[1] Assistant Professor, IBB Group of Colleges, Ranpur, Kota Rajasthan

[2] Assistant Professor, IBB Group of Colleges, Ranpur, Kota Rajasthan

be referred to as hard failures, and the second type as soft failures. The protocol should provide different mechanisms to tolerate the two types of failures. The objective of the present work is to design a checkpointing approach which is suitable for mobile computing environment.

3. SYSTEM MODEL

The mobile environment model used in this protocol contains both fixed and mobile hosts interconnected by a backbone network. The fixed hosts are called MSS and mobile hosts are connected to MSS by wireless links. A MSS is connected to another MSS by wired network. The static network provides reliable and sequenced delivery of messages between any two MSSs. Similary wireless link between MSS and MH ensures FIFO delivery of messages. An MH can directly communicate with MSS only if the MH is physically located in that MSS. A cell is a geographical area around MSS which can have many MH. An MH can freely move from one cell to another and change its geographical position. At any instant of time, an MH can belong to only one cell. If an MH does not leave its cell, then every message sent to it from local MSS would be received in sequence in which it is sent.

3.1 Algorithm Concept

We assume that the protocol maintains a unique checkpoint number counter, CkpNum, at each process to guarantee that the independently saved checkpoints verify the consistency property. Whenever the process creates a new checkpoint, the value of CkpNum is incremented and is piggybacked with every message. The consistency property is ensured if no process receives a message with a CkpNumi larger than the current local CkpNum. If CkpNumi is larger than the local CkpNum, the process creates a new checkpoint before delievering the message to the application. The recoverability property is guaranteed by logging at the sender all messages that might become in-transit. These are the messages that have not been acknowledged by the receivers at checkpoint time. The sender process also logs the send and receive sequence number counters. During normal operation, these counters are used by the communication layer to detect lost messages and duplicate messages due to retransmissions. After a failure, each process resends the logged messages. Duplicate messages are detected as they are during the normal operation.

3.2 Creation of a Global State

Whenever a mobile host moves out of the range of the cell or user turns off the network interface, it becomes disconnected. In a disconnected mode, the mobile host cannot access any information that is stored on a stable storage. Due to this reason, the protocol must be able to perform its duties correctly using local information. The protocol continues to save soft checkpoints to recover from soft failures. Two types of disconnections are considered. A Temporary disconnection allows the protocol to exchange few messages with stable storage just before the mobile host becomes isolated. Examples include the situations where communication layer informs the protocol when mobile host moves outside the range of cell or the boundary areas where signal strength becomes weaker. A permanent disconnection implies the case in which protocol is not able to exchange any messages with stable storage. Example includes when use unplugs the cable without turning off the application.

The creation of a new global state before disconnection is necessary for both the mobile host and the other hosts. This new global state is important because it prevents the rollback of work that was done while the mobile host was disconnected. If the new global state is not saved and another host fails after the disconnection, the application rolls back to the last global state that was stored (without warning the mobile host). The mobile host cooperates with the stable storage to create a new global state before disconnection. Just before the mobile host becomes isolated, the protocol sends to stable storage a request for checkpoint, and saves a new checkpoint of the process (hard or soft, depending on the network). Then the stable storage broadcasts the request to the other processes. Processes save their state as they receive the request. New global states can only be created before the mobile host detaches from the network if disconnections are orderly.

When the mobile host reconnects, the protocol sends a request to stable storage, asking for the current checkpoint number and the CN of the last hard global state. When the answer arrives, the protocol updates the local CN using the current checkpoint number. The protocol also creates a hard checkpoint if the mobile host has been isolated for a long time.

3.3 Working of the Algorithm

We illustrate the execution of the protocol with the help of following figure. This figure (Figure 1) represents the execution of three processes. Processes create their checkpoints at different instants, because timers are not synchronized. After saving its CkpNum checkpoint, process P1 sends message m1. When m1 arrives, process P3 is still in its CkpNum-1 checkpoint interval. To avoid a consistency problem, P3 first creates its CkpNum checkpoint, and then delivers m1. P3 also resets the timer for the next checkpoint. Message m2 is an in-transit message that has not been acknowledged when process P2 saves its CkpNum checkpoint. This message is logged in the checkpoint of P2. Message m3 is a normal message that indirectly resynchronizes the timer of process P2. It is possible to observe in the figure the effectiveness of the resynchronization mechanism.

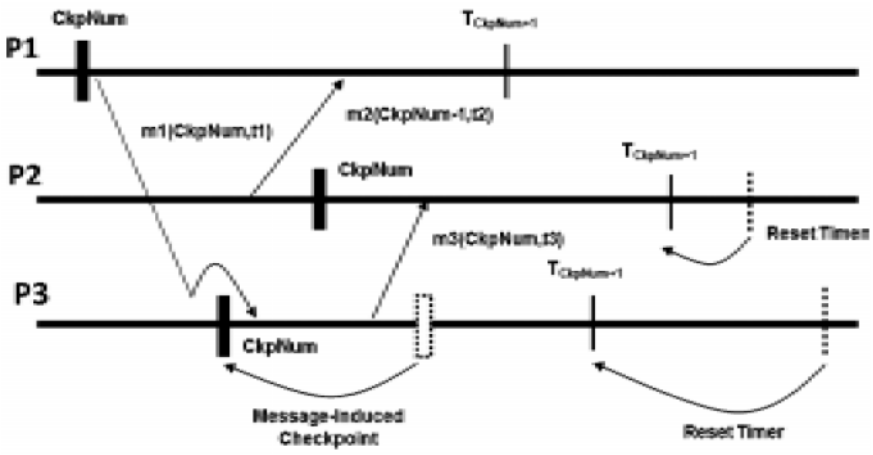

Figure 1: Time-based Soft Checkpointing

3.4 The Algorithm

Following is the pseudocode of the algorithm. The algorithm uses the following local variables–

// Si - Sender's Identifier

// CkpNumi - Current checkpoint number of the sender

// timeToCkpi - Time interval until next checkpoint

// msgi – Message contents

A . Message Receiving

receiveMsg (S_i, $CkpNum_i$, $timeTockp_i$, msg_i)

if ((CkpNum=$CkpNum_i$) and getTimeToCkp( ) > timeToCkpi))

resetTimer (timeToCkpi);

else if (CkpNum < $CkpNum_i$) {

CreateCkp ( );

resetTimer($timeToCkp_i$);

}

delieverMsgToApplication (msgi);

B. Application Process (At MH)

createCkp ( ) :

CkpNum : = CkpNum +1;

resetTimer (T);

if ((CkpNum mod maxsoft) = 0) sendCkpST (getState ( ));

else saveState (getState ( ), CkpNum);

C. Stable Storage (At MSS)

// The function arguments are same as in message receiving

receiveCkp (S_i, $CkpNum_i$, $timeTockp_i$, $state_i$)

saveState ($state_i$, $CkpNum_i$);

CkpNum : = max (CkpNum, $CkpNum_i$);

setFlag ($CkpNum_i$, S_i);

if (row ($CkpNum_i$) = 1) {

CkpHard : = CkpNumi;

garbageCollect (CkpHard);

}

The functions given above are used to create a new checkpoint. Function createCkp is called to save a new process state. It starts by incrementing the CkpNum, and then it resets the timer with the checkpoint period. Next, the function determines if the checkpoint should be saved locally or sent to stable storage. The function saveState stores the process state locally, and the function sendCkp sends the process state to stable storage. The function receiveCkp is called by the stable storage to store newly arrived checkpoints. It first writes the received state to the disk, and then updates the local checkpoint counter. Then, it determines if a new hard global state has been stored using a checkpoint table. The checkpoint table contains one row per CkpNum, and one column per process. The table entries are initialized to zero. An entry is set to one whenever the corresponding checkpoint is written to disk. The table only needs to keep one bit per entry, which means that it can be stored compactly. A new hard global state has been saved when all entries of a row are equal to one. The variable CkpHard keeps the checkpoint number of the new hard global state. The function garbageCollect removes all checkpoints with checkpoint numbers smaller than CkpHard.

3.5 Proof of Correctness

Theorem 1: The algorithm ensures a consistent global checkpoint

Proof: The consistency property is ensured if no process receives a message $CkpNum_i$ larger than current CkpNum. The process creates a new checkpoint before delievering the message to the application if $CkpNum_i$ is larger than the local CkpNum. All processes restart from an appropriate state because, if they decide to restart, they resume execution from a consistent state (the checkpointing algorithm takes a consistent set of checkpoints). We consider following three cases:

Case 1: For all MHs about to disconnect, the algorithm takes a checkpoint before the disconnection. Hence, its checkpoint is used as an initial checkpoint after a reconnection. Therefore, no inconsistent state occurs due to the disconnection.

Case 2: For all MHs in doze mode, the algorithm sends a request message to wake up them to take a checkpoint. Hence, all MHs in doze mode can not cause an inconsistent state.

Case 3: When a MH moves from a cell of MSS_i to another cell of MSS_j, MSS_j sends a hand-off message to MSS_i to obtain the MH's checkpoints. Hence, a consistent state is kept when a MH moves.

As discussion above, we prove that the algorithm ensures a consistent global checkpoint even if it is a special case.

Theorem 2: The algorithm rolls back all necessary processes to a global consistent state when failure occurs.

Proof: An orphaned message can cause an inconsistent state after a failure recovery. An orphan message M occurs when the states of MH_i and MH_j are rolled back to checkpoints $C_{i,a}$ and $C_{j,b}$, where a and b are checkpoint indices of MH_i and MH_j, and M is sent after $C_{i,a}$, but it is received before $C_{j,b}$. There are two cases depending on whether the message is sent before or after the failure. If it occurs before the failure, MH_i sends message M to MH_j after $C_{i,a}$, and MH_j receives M, and then takes $C_{j,b}$. Based on Section 3.4 part A we prove that recoverability property is guaranteed by logging at the sender all messages that might become in-transit. These are the messages that have not been acknowledged by the receivers at checkpoint time. The sender process also logs the send and receive sequence number counters. During normal operation, these counters are used by the communication layer to detect lost messages and duplicate messages due to retransmissions. After a failure, each process resends the logged messages. Duplicate messages are detected as they are during the normal operation.

4. COMPARISON WITH THE RELATED WORK

In this section we compare our work with Acharya and Badrinath [6] and Pardhan et al. [7] since our work is very closely related to their work. The protocol designed by Acharya and Badrinath [6] requires to create a new checkpoint whenever they receive a message after sending a message. Processes also have to create a checkpoint prior to disconnection. Pardhan et al. [7] proposed two uncoordinated protocols. The first protocol creates a checkpoint every time when a process receives a message. The second protocol creates checkpoints periodically and logs all messages received. Also, the two protocols proposed in [6] and [7] always assume hard failures. These two algorithms have the following good features:

1. Only those processes that have received some message after sending a message, take checkpoints during checkpointing [6] or when process receives a message [7] thereby reducing the number of checkpoints to be taken.
2. Reductions in the number of checkpoints help in the efficient use of the limited resources of mobile computing environment.
3. Uses minimum interaction (only once) between the initiator process and the system of n processes and there is no synchronization delay.

However, the algorithms have a limitation too. Consider a system of n process distributed system. Let, the cost of sending a checkpoint request message from initiator to a single process be C_i. Hence, the checkpoint request cost, incurred by a single execution of the checkpointing algorithm, would be $(n–1)C_i$. Thus, the checkpoint request cost, incurred by k executions of the checkpointing algorithm, would amount to $k(n–1)C_i$. Therefore, the checkpoint request overhead, for applications involving large number of processes and running for longer durations, increases exponentially.

In the present work, we have attempted to eliminate above problem by using timer. It is a well-known fact that the use of timer eliminates extra coordination messages [10]. A process takes checkpoint whenever its local timer expires. Moreover, only those processes take checkpoint, after expiry of their local timer, who have sent at least one message in the current checkpoint interval. Therefore, the number of processes taking checkpoint and, subsequently, the total number of checkpoints is significantly reduced. In addition, the use of timer removes need of the initiator process for sending the checkpointing request messages.

Our protocol creates a checkpoint whenever the local timer expires, and it only logs the unacknowledged messages at checkpoint time. Our protocol uses two types of checkpoints to recover from failure - soft checkpoints created and stored in MH to recover from soft failures and hard checkpoints created and stored at MSS to recover from permanent failures. Table 1 gives a comparison of our work on different parameters with the protocols proposed in [6] and [7].

Table 1
Comparison with the Related Work

Parameters	Acharya and Badrinath [6]	Pardhan et al. [7]	Our Protocol
Creation of checkpoint	When a new message is received after sending a message	When process receives message	When local timer expires
No. of checkpoint phases	1	1	2
Failure assumed	Hard	Hard	Hard and Soft
Adaptable	No	Depend on wireless bandwidth	Vary with QoS of network
Coordination Method	Message based	Message based	Timer Based
Checkpoint Latency	High	High	Low
Transmission Cost	High	High	Low
Recovery Time	High	High	Less
CPU Overhead	High	High	High
Additional Hardware	Not Required	Not Required	Additional processor is required on MH
Main Memory requirement	Low	Low	High
Reliability	Low	Low	High
Efficiency	Low	Low	High
Suitability	For Large Systems	For Large Systems	For Large and small systems

5. CONCLUSION

In our proposed approach, a have described a protocol that is able to save consistent recoverable global states. The process creates a new checkpoint whenever the local timer expires. The protocol stakes a soft checkpoint and saves it on the mobile host and later on before disconnection converts it to the hard checkpoint that is stored on MSS as soft checkpoint is less reliable. The protocol adapts its behavior to different types of networks by changing the number of soft checkpoints to be taken per hard checkpoint. When the mobile host is disconnected, the protocol creates soft checkpoints to recover from soft failures. The main features of our algorithm are: (1) it is non-blocking; (2) it is adaptive because it takes checkpointing decision on the basis of checkpoint sequence number; (3) it doesn't require tracking and computation of dependency information; (4) it doesn't require any control message because it uses timer to indirectly coordinate the creation of consistent global checkpoints and the local timers are not synchronized through control messages but by piggybacking control information on application messages.

REFERENCES

[1] C. Chowdhury, S. Neogy, "A Consistent Checkpointing-Recovery Protocol for Minimal number of Nodes in Mobile Computing System", in International Conference on High Performance Computing, pp. 599-611, 2007.

[2] Koo R. and Toueg S. (1987). "Checkpointing and Roll-back Recovery for Distributed Systems", IEEE Trans. on Software Engineering, 13(1), January, pp. 23-31.

[3] G. Cao and M. Singhal, "On Coordinated Checkpointing in Distributed Systems", IEEE Trans. Parallel and Distributed System, pp. 1213-1225, Dec. 1998.

[4] E.N. Elnozahy, D.B. Johnson, and W. Zwaenepoel, "The Performance of Consistent Checkpointing", Proc. 11th Symp. Reliable Distributed Systems, pp. 86-95, Oct. 1992.

[5] R. Prakash and M. Singhal, "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems", IEEE Trans. Parallel and Distributed Systems, pp. 1035-1048, Oct. 1996.

[6] Acharya A. and Badrinath B.R. "Checkpointing Distributed Applications on Mobile Computers". In Proceedings of the Third International Conference on Parallel and Distributed Information Systems, (Austin, Texas, Sep, 1994), pp. 73-80.

[7] Pradhan D.K., Krishna P., and Vaidya N.H. Recovery in Mobile Environments: Design and Trade-off Analysis". In Proceedings of the 26th International Symposium on Fault-Tolerant Computing, (Sendai, Japan, June 1996), IEEE, pp. 16-25.

[8] G. Cao and M. Singhal, "Mutable Checkpoints : A New Checkpointing Approach for Mobile Computing Systems", In Proceedings of the IEEE Trans. 12(2), pp. 157-172, Feb. 2001.

[9] Randell B. "System Structure for Software Fault Tolerance". IEEE Trans. Softw. Eng. SE-1, 2 (June 1975), 220-232.

[10] N. Neves, "Time-based Coordinated Checkpointing", Ph.D. Dissertation, UIUCDCS-R-98-2054, University of Illinois at Urbana-Champaign, 1998.