TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 20/49 pp110-115
Volume 12, Number S1, July 2007

Low-Overhead Non-Blocking Checkpointing Scheme for Mobile Computing Systems*

MEN Chaoguang (门朝光)[1,2], CAO Liujuan (曹刘娟)[1,**],
WANG Liwen (王立闻)[1], XU Zhenpeng (徐振朋)[1]

1. R & D Center of High Dependability Computing Technology, Harbin Engineering University, Harbin 150001, China;
2. National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China

Abstract: When applied to mobile computing systems, checkpoint protocols for distributed computing systems would face many new challenges, such as low wireless bandwidth, frequent disconnections, and lack of stable storage at mobile hosts. This paper proposes a novel checkpoint protocol to effectively reduce the coordinating overhead. By using a communication vector, only a few processes participate in the checkpointing event. During checkpointing, the scheme can save the time used to trace the dependency tree by sending checkpoint requests to dependent processes at once. In addition, processes are non-blocking in this scheme, since the inconsistency is resolved by the piggyback technique. Hence the unnecessary and orphan messages can be avoided. Compared with the traditional coordinated checkpoint approach, the proposed non-blocking algorithm obtains a minimal number of processes to take checkpoints. It also reduces the checkpoint latency, which brings less overhead to mobile host with limited resources.

Key words: mobile computing; fault tolerant; coordinated checkpoint; rollback recovery

Introduction

In recent years, many algorithms supporting distributed systems are modified to be suitable for mobile computing. However, it is well recognized that the previous checkpointing-recovery schemes are not appropriate in the mobile environments due to the unique characteristics of mobile networks and devices. Comparing with traditional distributed systems, mobile computing faces many new challenges such as low wireless bandwidth, frequent disconnection, and lack of stable storage at mobile hosts (MHs)[1,2]. Hence, the checkpoint protocols proposed for distributed systems have been well investigated[3,4], which cannot be directly applied to mobile computing.

Previously proposed checkpoint algorithms for mobile environment can be categorized into two groups: coordinated algorithm and uncoordinated algorithm. Most proposed coordinated checkpoint algorithms either force all the processes to take checkpoint or block participating processes while checkpointing, which may dramatically degrade the system performance. In uncoordinated algorithms, processes take checkpoints independently and no coordinated messages are involved. However, uncoordinated checkpoint algorithms may suffer from the domino effect and the rollback recovery algorithms are much more complicated.

Checkpoint algorithms for mobile computing environment have been well investigated[5-12]. In Ref. [6],

Received: 2007-02-01
* Supported by the Postdoctoral Science Foundation (No. 20060390461) and the Basic Research Foundation of Harbin Engineering University (Nos. HEUF040806, HEUFT05009, and HEUFP05020)
** To whom correspondence should be addressed.
E-mail: caoliujuan@126.com

the mutable checkpoint algorithm utilized two types of checkpoints to minimize the number of both synchronization messages and checkpoints. But it is pointed out the algorithm can cause inconsistencies in some situation. In Ref. [8], the authors proposed a time-based checkpoint protocol that tries to reduce the number of checkpoints. However, the two types of checkpoint protocols have not been taken into consideration for reducing the checkpoint latency. In Ref. [9], the authors used a "filtering" process at BS to reduce the checkpointing participants, and used a communication vector at the associated mobile support station (MSS) to significantly decrease the latency time. But the number of the checkpoints is quite large.

In this paper, we propose a low overhead non-blocking checkpointing scheme for mobile computing (LNCSM) to reduce both the coordinating overhead of mobile hosts and the number of the checkpoint participants. By using a communication vector maintained at the associated MSS, only a few processes participate in the checkpointing event. According to the information piggybacked, orphan message[13] and unnecessary checkpoint can be avoided. Hence our method brings less overhead to a mobile host with limited resources.

1 System Model

We assume that a set of MHs is running a distributed application that consists of N sequential processes denoted by $P_1,P_2,...,P_N$ running concurrently on MHs in the network. Each MH has limited memory. The only way in which the processes can communicate with each other is through message passing. Every process saves its local state into a stable storage to produce its local checkpoint. Later, a copy of the checkpoint will be sent to the associated MSS. Each checkpoint taken by a process is assigned a unique checkpoint sequence number (CSN). The i-th ($i\geqslant 0$) checkpoint of process P_k is assigned a sequence number i and denoted by $C_{k,i}$. The k-th checkpoint interval of a process denotes all the computation performed between its k-th and (k+1)-th checkpoints, including the k-th but not the (k+1)-th checkpoint.

Assume that only the MHs will fail due to some errors, and the associated MSSs can detect this failure. We distinguish two kinds of messages: computation messages and system messages. Computation messages are those sent for their underlying application purposes, while system messages are those sent for coordinating checkpoint purposes.

2 LNCSM Algorithm

2.1 Data structure

tnt_V_i: a vector that implies the direct dependency relationship.

V_i: a vector that implies the direct and indirect dependency relationship. It is used to save the information that which processes have taken new checkpoint. It is initialized to 0.

V_{ji}: a boolean variable. It is set to 1 when process P_i has received one or more messages from P_j since p_i's last checkpoint. Initially, all V_{ji} entries are 0 and the entry V_{ii} remains at 0 (no self-communication).

$CSN_i[j]$: an integer array. $CSN_i[j] = X$ means that process P_j takes X-th checkpoint that P_i expects. $CSN_i[i]$ is initialized to 0 in every process.

Trigger: a tuple (Pid, inum). Pid indicates the checkpointing initiator that triggered this node to take its latest checkpoint, inum indicates the CSN at node Pid when it takes its local checkpoint on initiating consistent checkpointing.

Com_state: a Boolean array. It is set to 1, if P_i has taken a forced checkpoint.

R_i: a Boolean variable. It is used to detect the termination of the checkpoint. If a process P_i takes a tentative checkpoint, it sets R_i=1, sends R_i=1 to the initiator. R_i is initialized to 0 in every process.

2.2 Basic idea

Assume that a process will not receive a checkpoint request associated with another initiator until the current executing one is completed. There is only one checkpoint per process stored in MH's stable storage and a copy of this checkpoint is also present in the associated MSS. The associated MSS for a process P_i maintains a corresponding communication vector, tnt_V_i of size N. Let tnt_V_i= $\{V_{1i}\ V_{2i}...V_{ni}\}$ that implies the direct dependency relationship. Simply V_{1i} is set to 1 in the vector while P_1 has sent message to P_i since P_i's last checkpoint. V_i is a vector that implies the direct and indirect dependency relationship. When process P_i initiates a checkpoint, the associated MSS of process P_i along with the associated MSS of process P_j such that V_{ji}=1 set the vector V_i= tnt_V_i ∪ tnt_V_j , tnt_V_i,

tnt_V_j are reset to 0 immediately. During the current checkpoint interval, V_i will be updated whenever there is any change in tnt_V_i , tnt_V_j. This technique can cope with the tardy message[10], this kind of message can induce new dependency relationship.

We use three types of checkpoints: tentative, forced, and permanent checkpoints. These processes on which the initiator depends should take tentative checkpoint, and forced checkpoint is used to avoid orphans. When a process takes a forced checkpoint, it does not send its dependency relationship to the initiator. The forced checkpoint should be transformed to a tentative checkpoint when the process that has taken the forced checkpoint receives request or discarded when the process receives committing message. Additionally, after receiving all the R_j=1 from P_j such that V_{ji}=1 according to the vector V_i, the initiator P_i broadcasts committing message to all processes in the system, piggybacking the information that which processes have taken checkpoints according to the associated vector V_i. The checkpointing processes are non-blocking in the scheme as the inconsistency is resolved by the piggyback technique.

2.3 LNCSM algorithm

(1) Initiating a checkpoint process

Each process running on an MH can initiate a checkpoint. In our consistent non-blocking algorithm, not all the processes participate in a checkpoint event. When a process P_i initiates a checkpointing, it takes a tentative checkpoint, increments CSN$_i$[i], saves its state in stable storage, transmits the copy of the checkpoint to its associated MSS. The associated MSS of P_i along with the associated MSS of process P_j such that V_{ji}=1 set V_i= tnt_V_i ∪ tnt_V_j , tnt_V_i and tnt_V_j are reset to 0 immediately. During the current checkpoint interval, V_i will be update whenever there is any change in tnt_V_i, tnt_V_j. The initiator sends a checkpoint request to each process P_j such that V_{ji} =1 in the vector V_i and resumes its computation. Each request carries the trigger of the initiator

(2) Reception of a checkpoint request message

When a process P_j receives a request from P_i, firstly, the associated MSS of P_j checks the vector tnt_V_j. In case that there is a process on which P_j depends, but P_i not depends, the associated MSS of P_i along with the associated MSS of P_j set V_i= V_i ∪ tnt_V_j, tnt_V_j is reset to 0 immediately. Then if detecting Com_state=1, P_j makes forced checkpoint tentative, sends R_j=1 to the associated MSS of the initiator. Otherwise P_j takes a tentative checkpoint, increments CSN$_i$[i], saves its local state to the stable storage, and sends the copy of the checkpoint to the associated MSS, then sends a reply R_j=1 to the initiator. P_j resumes its underlying computation.

(3) Sending and receiving a computation messages

When a process P_j in checkpointing sends a computation message to a process P_i, it piggybacks its CSN$_j$[j]. When process P_i receives a computation message m from P_j and P_i has not taken new checkpoint, firstly, the associated MSS of P_i sets the vector tnt_V_i that V_{ji}=1, then P_i compares CSN$_j$[j] with its local CSN$_i$[j]. If CSN$_j$[j]≤CSN$_i$[j], P_i delivers m directly. Otherwise, it implies that P_j has taken a checkpoint before sending m. P_i takes a forced checkpoint, increments CSN$_i$[i], sets Com_state=1, saves its local state to the stable storage, then delivers m. But P_i does not need to send the copy of the checkpoint to the associated MSS. Figure 1 is an example of sending and receiving a computation message.

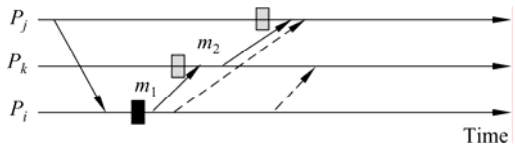

Fig. 1 An example of sending and receiving a computation message

In Fig. 1, solid line means transmitting computing message and dashed line means request message. As an initiator, the process P_i takes a checkpoint, then sends message m_1 to process P_k with CSN$_i$[i]=1. The associated MSS of P_k sets the vector tnt_V_k that V_{ik}=1. Detecting CSN$_k$[i]=0, CSN$_i$[i]＞CSN$_k$[i], P_k takes forced checkpoint before delivering m_1, increments CSN$_k$[k], sets Com_state=1, saves checkpoint in its stable storage, does not need to transmit the copy of the checkpoint to the associated MSS. Similar to P_k, P_j sets tnt_V_j that V_{kj}=1, takes forced checkpoint before delivering m_2. After receiving checkpoint request, P_j makes forced checkpoint tentative, sets Come_state=0, sends R_j=1 to the initiator. Since P_k is such process on which P_j depends, but P_i does not, V_i=V_i ∪ tnt_V_j, P_i sends request to P_k. After receiving request, P_k makes forced checkpoint tentative. The system is consistent.

(4) Termination of checkpointing process

Many non-blocking algorithms for mobile computing use the weights adding up to 1 to test the termination of a coordinated checkpoint collection[12]. Our method uses the vector V_i and R_j to test the global checkpoint termination. If the associated MSS of P_i has received all the R_j=1 from processes P_j such that V_{ji}=1, it concludes that all the processes on which the initiator depends have taken their tentative checkpoints. Then the initiator broadcasts committing message with V_i to all processes. On receiving the committing message, if a process has taken a tentative checkpoint, it makes tentative permanent. If a process P_j has taken a forced checkpoint, it discards the forced checkpoint and decreases $CSN_j[j]$. If a process P_j has not taken checkpoint, but a process P_k on which P_j depends has taken checkpoint, the V_{jk}=1 should be changed to V_{jk}=0 to avoid unnecessary checkpoint in the future. The system is consistent. All the vectors remained act to be the communication vectors of the next checkpoint interval.

2.4 An example

Figure 2 is an example of checkpointing execution. In which solid line means transmitting computing message and dashed line means request message. There are six processes in the system. Suppose that P_4 is running on an MH. At time t_0, the system is consistent, and all the corresponding communication vectors are reset to 0. At time t_1, processes P_3 and P_5 have send messages to P_4. The communication vector tnt_V_4 maintained at the associated MSS updates to reflect this fact setting V_{34}=1 and V_{54}=1. Similar comments are true when the process P_1 sends message to P_5 and P_5 sends message to P_6. At time t_1, all the direct dependency relationship has been taken a record. Then P_4 as initiator takes checkpoint $C_{4,1}$ and sends request with trigger to the process P_i such that V_{i4}=1. So P_4 sends request to P_1, P_3, and P_5. After taking checkpoint $C_{3,1}$, P_3 sends m_4 to P_2 with $CSN_3[3]$=1. Due to $CSN_2[3]$=0, $CSN_2[3]<CSN_3[3]$, P_2 takes a forced checkpoint $C_{2,1}$, before delivering m_4, sets tnt_V_2={0,0,1,0,0,0}. Due to $CSN_1[2]$=0, $CSN_2[2]$=1; P_1 takes a forced checkpoint $C_{1,1}$ before delivering m_5, sets tnt_V_1={0,1,0,0,0,0}. Because the vector V_4 contains V_{24}=0, the associated MSS of P_4 sets V_4= $V_4 \cup$ tnt_V_1, later, sends request to P_2. After receiving request, P_1 makes forced checkpoint $C_{1,1}$ tentative, sends R_1=1 to P_4. Similarly, after receiving request, P_2 makes forced checkpoint tentative too. After receiving R_1=1, R_2=1, R_3=1, R_5=1, P_4 propagates the committing message, makes tentative checkpoint permanent. To avoid unnecessary checkpoint in the future, the vector tnt_V_6 is set to 0. The system is consistent.

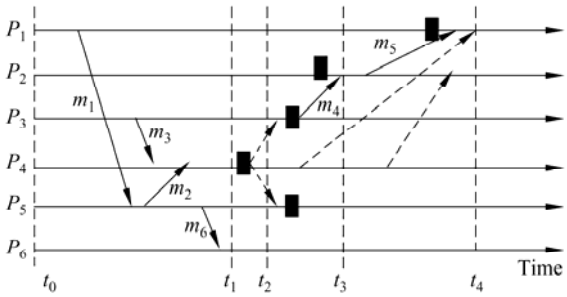

Fig. 2 An example of checkpointing execution

t_1: tnt_V_1= {0,0,0,0,0,0};
tnt_V_2= {0,0,0,0,0,0};
tnt_V_3= {0,0,0,0,0,0};
tnt_V_4= {0,0,1,0,1,0};
tnt_V_5= {1,0,0,0,0,0};
tnt_V_6= {0,0,0,0,1,0}.

t_2: tnt_V_1= {0,0,0,0,0,0};
tnt_V_2= {0,0,0,0,0,0};
tnt_V_3= {0,0,0,0,0,0};
tnt_V_4= {0,0,0,0,0,0}; V_4={1,0,1,0,1,0};
tnt_V_5= {0,0,0,0,0,0};
tnt_V_6= {0,0,0,0,1,0}.

t_3: tnt_V_1= {0,0,0,0,0,0};
tnt_V_2= {0,0,1,0,0,0};
tnt_V_3= {0,0,0,0,0,0};
tnt_V_4= {0,0,0,0,0,0}; V_4={1,0,1,0,1,0};
tnt_V_5= {0,0,0,0,0,0};
tnt_V_6= {0,0,0,0,1,0}.

t_4: tnt_V_1= {0,0,0,0,0,0};
tnt_V_2= {0,0,0,0,0,0};
tnt_V_3= {0,0,0,0,0,0};
tnt_V_4= {0,0,0,0,0,0}; V_4={1,0,1,0,1,1};
tnt_V_5= {0,0,0,0,0,0};
tnt_V_6= {0,0,0,0,0,0}.

2.5 Rollback recovery

When a failure occurs, the associated MSS can detect

this failure and the MSS notifies the failure to initial process P_i, then P_i notifies the failure to all the processes P_j such that V_{ji}=1 contained in vector V_i. All of them will be rolled back to the latest recovery line. Suppose that in Fig. 2, P_4 fails at a time between t_2 and t_3, the associated MSS will recognize the failure and notify this to processes on which it depends, i.e. P_3, P_5, and P_1. Other processes continue underlying computation normally. The system will roll back to the state at time t_0. Suppose that the failure occurs after time t_4. After a finite time period, the process P_4 recovers from the failure, when coming back to the network, P_4 gets its saved checkpoint from the associated MSS, and the MSS will send a "rollback" message to P_3, P_5, P_1, and P_2 according to V_4 to load the last saved checkpoint (stored at its stable storage) to start. So the non-fail processes do not need to load the checkpoints from the associate MSSs, which can reduce the overhead of the mobile computing system.

3 Correctness Proof

Theorem The algorithm creates a consistent global checkpoint.

Proof Assume there is an inconsistent after checkpointing. There must be at last one message m sent from P_i to P_j such that P_j saves the event of delivering m and P_i does not save the event of sending m. Because P_j takes a checkpoint, there are two cases:

Case 1 P_j is the initiator. Because m is sent from P_i to P_j, V_{ij}=1 must have been recorded in vector V_j. The associated MSS of process P_j will send request to P_i. P_i must take checkpoint. The system is consistent.

Case 2 P_j depends on initiator. Because m is sent from P_i to P_j, P_j takes a checkpoint. If P_i is not the process on which the initiator depends, the associated MSS of process P_j will send tnt_V_j to the initiator. The initiator will send request with trigger to process P_i. P_i must take checkpoint. If P_i is the process on which the initiator depends, P_i must take checkpoint too. The forced checkpoint is used to avoid orphan message and can not induce inconsistent. Hence the system is consistent.

4 Comparison with Other Algorithm

Many schemes have been proposed for mobile computing system. In Ref. [10], it forces few processes to take checkpoint, but the algorithm is blocking during checkpointing. Blocking algorithm may dramatically degrade the system performance. In Ref. [9], the checkpointing process is non-blocking, but the authors require all the processes both having sent message to initiator and having received message from initiator to take checkpoint, even though many of them may not be necessary. It causes the fact that the number of the coordinated messages is too large. Additionally, in some ad hoc algorithms, they do not take into consideration reducing the checkpoint latency from the time a process initiates a checkpoint request to the time the global checkpointing process completes.

Our algorithm requires only one checkpoint in a process, and the checkpoint can be saved in the associated MH's stable storage. When a failure occurs, the non-failure processes don not need to load the checkpoints from the associated MSS, which guarantees the less overhead on error recovery. Through dealing with computation messages, the algorithm can reduce the unnecessary coordinate communication information in the limited-bandwidth wireless network efficiently.

Since our method uses the communication vector and the piggyback technique, it only forces a few process to take checkpoints and to rollback in an error recovery and significantly reduces the latency associated with checkpoint request propagation, compared to traditional coordinated checkpoint approach.

5 Conclusions

In this paper we present a low-overhead non-blocking scheme for mobile computing system, which reduces the number of checkpoints and coordinating messages. Reducing such coordination overhead is important in mobile computing due to its limited bandwidth and limited resources of wireless network. Our protocol also takes advantage of the piggyback technique and the communication vector with reducing the checkpoint latency. Compared with traditional protocols, our scheme performs excellently in the aspect of minimizing the coordinating overhead, and in the aspect of minimizing the number of checkpoints.

References

[1] Men Chaoguang, Zuo D echeng, Yang Xiaozong. A new adaptive checkpointing strategy mobile computing.

Chinese Journal of Electronics, 2005, **14**(1): 15-20.

[2] Elnozahy M, Alvisi L, Wang Yimin, Johnson D B. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 2002, **34**(3): 375-408.

[3] Cao Guohong, Singhal M. On coordinated checkpointing in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 1998, **9**(12): 1213-1225.

[4] Koo R, Toueg S. Checkpointing and roll-back recovery for distributed systems. *IEEE Transactions on Software Engineering*, 1987, **SE-13**(1): 23-31.

[5] Cao Guohong, Singhal M. Checkpointing with mutable checkpoints. *Theoretical Computer Science*, 2003, **290**: 1127-1148.

[6] Cao Guohong, Singhal M. Mutable checkpoints: A new checkpointing approach for mobile computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 2001, **12**(2): 157-172.

[7] Park T, Woo N H Y. An efficient optimistic message lLogging scheme for recoverable mobile computing systems. *IEEE Transactions on Mobile Computing*, 2002, **1**(4): 265-277.

[8] Lin C Y, Wang S C. A low-overhead checkpointing protocol for mobile computing systems. In: Proceedings of the 2002 Pacific Rim International Symposium on Dependable. Tsukuba City, Ibaraki, Japan, 2002: 37-44.

[9] Ahmed R, Khaliq A. A low-overhead checkpointing protocol for mobile network. In: Canadian Conference on Electrical and Computer Engineering. Montreal, Que., Canada, 2003: 1779-1782.

[10] Li G, Shu L. A low-latency checkpointing scheme for mobile computing systems. In: Proceedings of the 29th Annual International Computing Software and Applications Conference. Edinburgh, UK, 2005: 491-496.

[11] Ni Weigang, Vrbsky S V, Ray S. Low-cost coordinated nonblocking checkpointing in mobile computing systems. In: Proceedings of the Eighth IEEE International Symposium on Computers and Communication. Kemer-Antalya, Turkey, 2003: 1427-1434.

[12] Manivannan D, Singhal M. A low-overhead recovery technique using quasi-synchronous checkpointing. In: Proc. 16th Int. Conf. on Distributed Computing System. Hong Kong, 1996: 100-107.

[13] Men Chaoguang, Wang Nianbin, Zhao Yunlong. Using computing checkpoint implement efficient coordinated checkpointing. *Chinese Journal of Electronics*, 2006, **15**(2): 193-196.