一种面向移动计算的低代价透明检查点恢复协议*

李庆华+, 蒋廷耀, 张红君

(华中科技大学 计算机科学与技术学院,湖北 武汉 430074)

A Transparent Low-Cost Recovery Protocol for Mobile-to-Mobile Communication

LI Qing-Hua+, JIANG Ting-Yao, ZHANG Hong-Jun

(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

+ Corresponding author: Phn: +86-27-87544471, Fax: +86-27-87544471, E-mail: liqh@263.net, http://www.nhpcc.hust.edu.cn

Received 2002-12-16; Accepted 2003-11-10

Li QH, Jiang TY, Zhang HJ. A transparent low-cost recovery protocol for mobile-to-mobile communication. *Journal of Software*, 2005,16(1):135−144. http://www.jos.org.cn/1000-9825/16/135.htm

Abstract: Mobile computing brings new challenges and requirements for checkpointing and recovery protocol. Existing checkpointing-only schemes can not guarantee the independent recovery through creating global consistent checkpoints. Message logging schemes based on mobile-MSS-mobile communication that exchanges messages among mobile hosts may incur large contention on the wireless network and high latency for message transmission relative to the direct mobile host to mobile host (m-m) communication. This paper presents a novel recovery protocol for m-m communication, in which two key problems, message order and duplicate message, are effectively solved. A proof of the protocol correctness is also given. Finally, simulation results indicate that the performance of the proposed approach is better than that of the traditional approaches in terms of fail-free and recovery overhead.

Key words: mobile computing; checkpoint; message logging; rollback recovery

摘 要: 移动计算系统中的检查点恢复协议面临着许多与传统分布式系统所不同的问题.在目前已出现的支持移动计算的检查点恢复机制中,基于建立全局一致的检查点的方法不能确保错误的独立恢复;基于 m-MSS-m 通信的消息日志方法其移动站之间交换的消息需通过移动基站的转发.提出了一种基于消息日志的支持移动站之间直接通信(m-m)的容错协议并给出了相应的算法及正确性证明.与 m-MSS-m 通信相比,m-m 通信有利于降低信道冲突;减少消息传递延迟.仿真结果表明,所设计的协议比传统协议具有更小的无错误状态下引入负载和错误恢复时间.

关键词: 移动计算;检查点;消息日志;回滚恢复

中图法分类号: TP393 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60273075 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.863-306-11-01-06 (国家高技术研究发展计划(863))

LI Qing-Hua was born in 1940. He is a professor and doctoral supervisor at School of Computer Science and Technology, Huazhong University of Science and Technology. His research areas include mobile computing and grid computing. **JIANG Ting-Yao** was born in 1969. He is a Ph.D. candidate at School of Computer Science and Technology, Huazhong University of Science and Technology. His research area is mobile computing. **ZHANG Hong-Jun** was born in 1964. She is a Ph.D. candidate at School of Computer Science and Technology, Huazhong University of Science and Technology. His research area is multi-agent.

1 Introduction

Recent wireless-LAN and personal-computer technology have made mobile computing realizable. A mobile computing system is a distributed system consisting of mobile hosts(MHs) that can move from one location to another and mobile support hosts(MSSs). MSSs are interconnected by a wired high speed network and their physical locations do not change. The geographical area covered by a wireless interface is called a cell. The mobile network is divided into multiple wireless cells. MHs keep moving from one cell to another. An MH communicates with another MH in another cell only through the MSS. Mobile computing raises many new issues: The bandwidth of a wireless LAN is lower than that of a wired LAN. Thus, the communication overhead must be minimized. The mobile hosts do not have so much capacity of the battery that it can not communicate with the other hosts for a long time and can not store very large data into its local storage to save energy. Handoffs and frequent disconnection because of lacking of the battery power are at large. That an MH leaves a cell and joins another cell immediately is called a handoff. Application/computation processes are required to continue their computation even when handoffs and disconnection occur.

Considering the fact that the MHs are vulnerable to failure, it is very necessary for a mobile computing system to be equipped with a checkpointing recovery facility. The state of a process is periodically saved on a stable storage, which is called a checkpoint. When failure occurs, the system rollbacks to restart its execution from the checkpoint and the messages that have been sent but not received are restored. Many checkpointing recovery protocols are reported[1−19]. However they are expensive and not adequate for mobile computing. The past protocols can be mainly divided into two categories: checkpointing-only and message logging. Checkpointing-only protocols have a common problem in rollback, which is that they need the exchange of extra synchronous messages. Recent work has shown that message logging has a decided advantage in recovery for mobile computing[1]. However message logging schemes are typically based on m-MSS-m communication. Exchanged messages among MHs in a same cell are retransmitted by the MSS, which increases the contention possibility and the latency for message transmission.

This paper presents a novel recovery protocol that can support m-m communication, which exchanges the message directly and is more suitable for the features of low bandwidth, fragile connection, and frequent failures. The rest of this paper is organized as follows: the related work is provided in section 2. Section 3 presents the system model. Section 4 is the proposed protocol description. Simulation results are given in section 5. Finally, section 6 gives our conclusions.

2 Related Work

Blocking coordinated checkpointing and recovery protocols[2−5] and nonblocking checkpointing schemes[6,7] have been extensively studied. They require the processes in system to synchronize their checkpointing or recovery activities in order to reach a global consistent state so that the reception of messages is recorded only if the corresponding sending has been recorded. Fixed hosts take consistent checkpoints by using synchronous protocols with the help of a high speed wired network and adequate stable storages. However, any kind of synchronization or coordination among MHs is not suitable due to the low network bandwidth, frequent disconnection, and insufficient storages.

References [8−10] discuss a synchronous checkpointing protocol for mobile hosts. The processes coordinate their checkpointing actions in such a way that the set of local checkpoints taken is consistent. Whenever a process requests to take a checkpoint, a set of processes must be checked and some of them may also need to take their checkpoints in order to preserve consistency. However, they are not adequate, because of the large number of

coordination messages exchanged between the MHs and processes participating in the coordination.

Asynchronous checkpointing[11] takes checkpoint periodically without any coordination with others. To recover from a failure, a process communicates with other processes to determine if their local states are causally related. However, this approach may suffer from the domino effect.

Communication-pattern based checkpointing[12] allows a process to take consistent checkpoints independently. The MHs may have to transfer a checkpoint with every outgoing message. In the worst case, the low bandwidth wireless network can not afford it. Communication-induced checkpointing[13] is difficult for all MHs to achieve synchronous realtime clock since message transmission delay is diverse and time-variant.

When the recovery is concerned, checkpointing only schemes may cause recursive rollbacks because of the livelock problem[2]. On way to guarantee the asynchronous recovery is the message logging in addition to the checkpointing. Message logging protocols are classified into three categories: pessimistic, optimistic, and causal. Causal protocols[14,15] require a large size of log space and also a large amount of dependency information to be carried in a message, which can be a serious drawback in the mobile environment. Pessimistic protocols[16] potentially block a process for each message it receives. This can slow down the throughput of the processes even when no process ever crashes. Optimistic protocols[17,18] , with a less message overhead than that of pessimistic, take the small risk of creating orphans and suffer from the transfer communication overhead. Rao compared the cost of recovery for different message logging approaches[19].

In China, some scholars have done many significant works about checkpoint and recovery[20−23].

Existing message logging schemes based on m-MSS-m communication are faster in recovery than that of the checkpointing only. However m-m communication benefits from reducing the contention on the wireless network, decreasing the latency for message transmission and increasing the throughput of MSSs. Thus, it is important to develop a new checkpointing recovery protocol for m-m mobile computing.

3 System Model

A mobile computing system $\Omega=\langle \upsilon, \tau \rangle$ consists of a set $\upsilon_s=\{mss_1,mss_2,\ldots,mss_n\}$ of MSSs, a set $\upsilon_h=\{mh_1, mh_2,\ldots,mh_m\}$ of MHs, and a set $\tau \subseteq \upsilon^2$ of communication channels. The network is divided into multiple cells. Each cell is supported by an MSS. The mh_i communicates directly with mh_j in the same cell and yet through υ_s with mh_k in another cell. This is realized by using wireless LAN protocols such as IEEE802.11, which is inherently broadcast-based. Message transmitted from a MH is received by a destination MH including the MSS in a same cell.

Distributed computation in a mobile computing environment is performed by a set of processes running concurrently on Ω(in the later section, the computation process run at mh_i is denoted by p_i). The computation is event-driven, and a message sending or a message receipt is called an event. Let e_i^x denote the x-th event of the process p_i. Let H be the set of all the events produced by a distributed computation. This computation is modeled by the partially order set $\hat{H}=(H,\xrightarrow{hb})$, where $\xrightarrow{hb}$ denotes the happen before relation[24]:

$$e_i^x \xrightarrow{hb} e_j^y \Leftrightarrow \begin{cases} i=j \wedge x \le y \\ \vee \exists m: e_i^x = send(m) \wedge e_j^y = receive(m) \\ \vee \exists k,l: e_i^x \xrightarrow{hb} e_k^l \wedge e_k^l \xrightarrow{hb} e_j^y \end{cases}$$

In addition to satisfying the happen before relation, the computation is also dependent on the states of the process, because a recovery process may not produce the same run upon recovery even if the same set of messages are sent to it since they may not be redelivered in the same sequence as previous failure. Process, p_i, experiences a sequence of state transits during its execution. Let r_i^m denote the m-th message receipt event of process p_i. The m-th state of p_i, s_i^m, indicates the sequence of the event generated between r_i^{m-1} and r_i^m (including r_i^m), where $m>0$

and r_i^0 is the initial event. Let p_i's state $[s_i^0, s_i^1, ..., s_i^m]$ represent the sequence of all states up to s_i^m.

In our model, a reliable message delivery is assumed, and the message transmission delay is assumed to be finite but arbitrary. The links abide by the FIFO communication. MHs are fragile but MSSs are reliable. Processes are failure stopped, and all failures result in halting a failed process immediately. The computation is assumed to follow the piece wise deterministic model.

4 The Proposed Recovery Protocols

The proposed recovery is based on the independent checkpoint, message logging and asynchronous rollback-recovery. We implement this approach with the help of the agent technique. A group of agent processes are in charge of taking checkpoint, message sending, receiving, and rollbacking, which is transparent to the computation.

4.1 Data structure

The following data structure is adopted in the proposed recovery algorithm:

sent[] and *recv*[]: two integer array variables managed by each MH in the volatile storage. For process p_i, *sent*[*j*] and *recv*[*j*] are equal to the sequence number of the computation messages sent to and received from process p_j, respectively.

rec_sum: an integer variable managed by each MH in the volatile storage. It records the total number of the messages that each MH received.

sent_fini[]: an integer vector maintained by each MH in its stable storage. The message number that p_i has ever sent to p_j is stored in *sent_fini*[*j*].

rec_num: an integer value in each MH's stable storage. It is used to record the total number of the messages that each MH has ever received.

rec_ord[]: a tuple (*pid,inum*) vector maintained by each MH in its stable storage. We use it to keep track of a message's arrival order. The *pid* means the process Id that sends the message, and the *inum* indicates the sequence number of the message receipt. For example, for process p_1, *rec_ord*[5]=(2,3) indicates that the 5-th message received by p_1 is the 3-th message sent to p_1 by p_2.

4.2 Independent checkpoint and message logging

Each mobile host periodically takes a checkpoint and the time interval between two consecutive checkpoints is determined by itself. The checkpoint with the related *send*[], *recv*[] and *rec_sum* is transmitted to a MSS, say mss_p, to be saved into mss_p's stable storage. The message log is managed by mss_p. Each message delivered to mh_i by mh_j is also received by mss_p, what mss_p needs to do is to log the message but not to transfer it when rollback or when mh_i and mh_j are not in a same cell. When a new checkpoint is taken, the previous message log and the checkpoint are discarded.

Two special issues must be properly solved, shown in Fig.1, which never occurrs in m-MSS-m communication network.

The first problem is the order of message receipt in recovery. It can be seen from Fig.1 that the order of messages m_1 and m_2 received by mh_i is m_1, m_2, but the order reaching mss_p is m_2, m_1. Thus when rollback happens after a failure occurrs in mh_i, mss_p can not confirm the arrival order of m_1, m_2 in mh_i. In other words, it is impossible for mss_p to store messages into a message log in the same order as mh_i does by only receiving them from a broadcasted wireless LAN protocol.

One idea to get a correct order is that mh_i stores messages in a volatile storage temporarily and transmits them to mss_p for storing into a stable storage. However mh_i does not always have enough storage and this will increase

communication overhead. Another idea introduced by Ref.[25] is to store the messages in mss_p without order. When mh_i needs to send a computation message, say m_4, it attaches the message arrival order to m_4. However this approach may not work correctly, such that a failure occurrs in mh_i before it sends message m_4. As a result, the order information may be lost. In addition, extra order information appended to every message incurs the communication overhead.

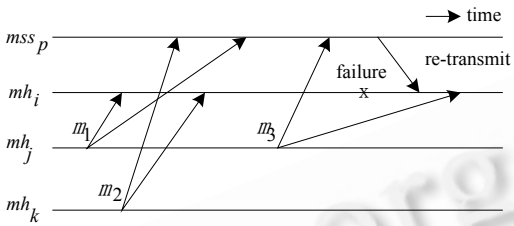

Fig.1 Message order and duplicate message

The second problem different from that in m-MSS-m network is that the delayed message may result in duplicate message received in failed hosts after rollback. See Fig.1, mh_j sends a message m_3 to mh_i, and m_3 reaches mss_p early because of the arbitrary latency time of message transmission. Then mh_i incurs a failure and that m_3 does not come yet. During the recovery, both the re-transmitted message from mss_p and the delayed message from mh_j will be received by mh_i. One of the two messages must be discarded, otherwise a duplicate message will appear. In the proposed protocols, mss_p simply logs the messages according to their arrival order, without transferring them and considering their arrival order in MHs. The work of identifying a correct sequence or justifying a duplicate message is done by mh_i itself.

Considering the problem of handling handoff and disconnection, if mh_i intends to be disconnected from the network, it needs to take a checkpoint. If a handoff takes place at mh_i, it first disconnects from the old MSS and then connects to the new MSS. When a handoff happens, checkpointing and message logging schemes may be one of the logging-pessimistic, Loging-lazy and Loging-trickle schemes introduced in Ref.[1], on which this paper does not focus.

4.3 Independent recovery

We design a sending agent process and a receiving agent process running in each MH to answer for each message sending and receiving of computation processes, illustrated in Fig.2.

When a computation process p_i need to send a message to p_j, it delivers the message to the sending agent process. If the message is its first sending($sent\_fini[j]<sent[j]$), the agent process sends it out, embedded with the sequence number of computation message sent to p_j, otherwise it is ignored. After receiving a computation message transmitted to p_i, the receiving agent process needs to identify not only whether the message is duplicate but also the retransmitted message's order. If there exists no failure, there is no duplicate message because MSS does not need to transfer the message in the same cell. In the case of failure, the agent process must deliver the messages to the computation process in the same receipt sequence as that of its pre-failure. We introduce an integer vector *rec_ord*[] to keep track of the order of the message (not message itself) delivered to the computation process. After receiving a message, the order is piggied back to the *rec_ord*[], see Fig.2. The order information is stored in a local stable storage, thus it can not be lost. Moreover no excess information attached to the message decreases the communication overhead.

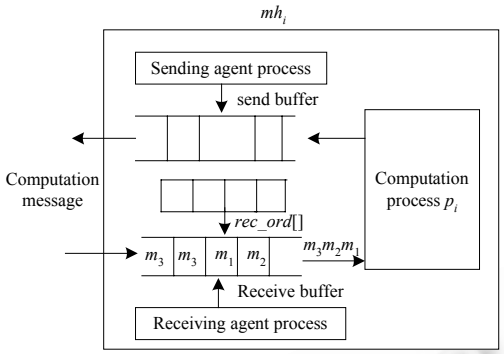

Fig.2 Message sending and receipt

Assume p_i fails or needs to reconnect, the rollback agent process sends a rollback request message to the mss_p. The mss_p responds by sending checkpoint, then p_i restores from the checkpoint and restarts. Sequentially the MSS replays the logged messages. The pseudocode for the algorithm is presented as follows.

Checkpoint agent process at mh_i, act periodically or when a disconnection occurs
take a checkpoint for computation process;
send(checkpoint, sent[], recv[], rec_sum) to mssp;
rollback agent process at mh_i, act when a failure occurs in mh_i or when a reconnection happens
send rollback request to mssp;
receive(checkpoint, *sent*[], *recv*[], *rec_sum*);
p_i restarts from checkpoint;
rollback agent process at mss_p, act when receiving rollback request of mh_i
send(checkpoint, *sent*[], *recv*[], *rec_sum*) to mh_i;
send the logged message if *sent*[*i*] in it is originally sent by mh_j
recv[*j*] when rollback is not over;
sending agent process at mh_i, act when p_i attempts to send a message to p_j
sent[*j*]++;
if (*sent_fini*[*j*]<*sent*[*j*])
{send(*sent*[*j*], message); *sent_fini*[*j*]++;}
receiving agent process at mh_i, act when receiving a message transmitted to p_i
if (*rec_sum*<>*rec_num*)
a: for every message in the receiving buffer
/**pid_m* is the ID of computation process that sends this message*/
if (*sent*[*i*] in a message =*rec_ord*[*rec_sum*+1].*inum* and *pid*_m=*rec_ord*[*rec_sum*+1].*pid*)
{deliver the message to p_i; *rec_sum*++; *recv*[*pid_m*]++; goto a}
if (*rec_sum*=*rec_num*)
if during rollback then send(rollback is over) to mss_p;
b: for every message in buffer
{for(*n*= 1; *rec_sum*; *n*++)
if (*sent*[*i*] in a message =*rec_ord*[*n*].*inum* and *pid_m*=*rec_ord*[*n*].*pid*)
the message is ignored and goto b; /*duplicate message*/
rec_sum++; *recv*[*pid_m*]++; *rec_num*++;

rec_ord[*rec_sum*]=(*pid_m*, *sent*[*i*] in the message);
deliver message to p_i;
}

4.4 Proof of correctness

The following definitions are based on the mobile computing system model described in section 3.

Definition 1. *Depend*(*m*) is a set containing all computation processes whose state reflects the delivery of message *m*. *Depend*(*m*) includes the destination process of *m* and any process whose message receipt events happen causally after the receipt of m . The *m.dest* is the destination process's Id of *m*. Formally

$$Depend(m)=\left\{p_j, j\in \upsilon_h \middle| j=m.dest \wedge r_j^m \vee (r_{m.dest}^m \xrightarrow{hb} r_j)\right\}.$$

Definition 2. *Memory*(*m*) is a set including all processes that have a copy of *m* in volatile memory. Process $p_{m.dest}$ is a member of *Memory*(*m*) after it receives *m*.

Definition 3. Process p_i is an orphan of *C* if p_i itself does not fail and p_i's state depends causally on the delivery of *m*, where *C* is a set of the failed processes. Formally:

$$Orphan(C)=\left\{p_i \middle| p_i \in \overline{C} \wedge \exists m:((p_i \in Depend(m)) \wedge (Memory(m)\subseteq C))\right\}.$$

Lemma 1. The proposed protocol guarantees that no set *C* of a failed process creates orphan processes.

Proof. Under the condition of reliable communication and MSSs, every computation message that is sent is safely logged in a stable storage, which means message *m* can be restored if it is lost.

Thus $\forall m:(Depend(m)\subseteq \{p_{m.dest}\}\subseteq Memory(m))$ holds,

then $\forall m:((Memory(m)\subseteq C)\Rightarrow(Depend(m)\subseteq C))\Leftrightarrow \overline{orphan(C)}$ holds.

Hence, no orphan process of *C* is created.

Lemma 2. If a process p_i is failed, its state can be reconstructed independently.

Proof. Let p_i's state be $[s_i^0, s_i^1, ..., s_i^l]$ before failure, which indicates events $e_i^0, ..., e_i^{x-1}, e_i^x, ..., e_i^y$, where l≤*y*, e_i^x is the first event from the last checkpoint and e_i^y is the last event before failure. After a failure, p_i should rollback and replay all of the events that start events $e_i^x, e^{x+1}{}_i, ..., e_i^y$.

During recovery, p_i "sends" messages following the execution step itself and replays the sending event e_i in the same sequence as pre-failure, but they aren't indeed delivered to other computation processes due to the fact that *sent_fini*[] is larger than the corresponding *sent*[]. On the other hand, message receipt is dealt with by the following cases:

Case 1: Messages retransmitted by MSS in a wrong order are delivered to p_i according to the receipt order of the pre-failure stored in *rec_ord*[], which is responded by the iteration a in receiving agent process.

Case 2: The duplicate message is discarded. Once a message is delivered to p_i, its *sent*[] in the message is a recorder in *rec_ord*[]. The for-cycle in iteration b takes charge of identifying whether a message is a duplicate one or not.

Case 3: During recovery, if other processes send messages to p_i, the proposed algorithm guarantees these messages are delivered to p_i until all message receipt events $r_i^0, ..., r_i^l$ are complete(*rec_sum*=*rec_num*).

Because all messages sent and the receipt events $e_i^x, e^{x+1}{}_i, ..., e_i^y$ are replayed, the p_i's state is reconstructed.

Theorem 1. The proposed protocol enables the system to be recovered to a global consistent state in the case

of f(f≥1) concurrent failures.

Proof. A global state of the system is composed of a set of local states of the processes in the mobile computing system. The theorem is true obviously from Lemma 1 and Lemma 2, i.e. the failed processes can independently be recovered but without creating any orphan process.

5 Performance Evaluation

The performance metrics concerned in this paper are the fail-free overhead and recovery overhead. A mobile computing environment is simulated, which consists of 15 MHs and, for simplicity, only one MSS. The bandwidth is 2 MB. The lengthes of computation message and system message are 1 KB and 50 bytes respectively. The size of checkpoint is 2 MB and the disk bandwidth is 1.7 MB. The checkpoint interval is 100s. The failure rate of a process is 10^{-2}, following a poisson process. The message sending and receiving are randomly. An ideal checkpointing-only, optimistic message logging and the proposed protocol scheme are brought into comparison.

The fail-free overhead is the system's execution time when it finishes the specific number of message sending events without failures. The measured time when the number of the finished events is 450, 900, 1500 and 1800 respectively and the message sending rate r is 0.1 is given in Fig.3. From the fail-free overhead data we can see that the proposed approach has the least time in every case due to the fact that there are no need for the coordination overhead in the checkpoint only protocol, no need for the delay of logging messages in the pessimistic log protocol which slows down the throughput of the MSS even when no failure, and no need for the forwarding message latency in the optimistic approach which strengthens contention probability on the wireless network.

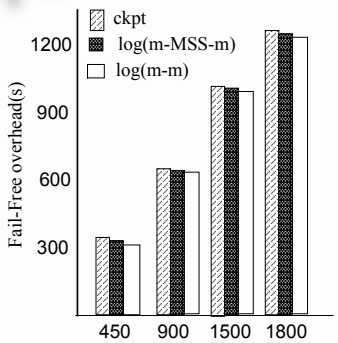

Fig.3 Fail-Free overhead

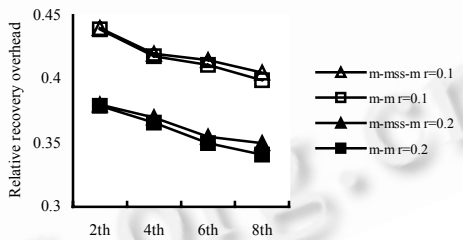

Fig.4 Relative recovery overhead

Recovery overhead is the time obtained by measuring the time for a process to read the checkpoint and proceed to a specific execution point. We choose the point to be after the 2nd, 4th, 6th, and 8th sending events of the last checkpoint. The message sending raters in our experiments are 0.2 and 0.1. For a distinct indication of the results, the relative values of the recovery overhead showed in Fig.4 are adopted, which are the ratio of the recovery overhead using m-MSS-m log to that using checkpointing-only and the ratio of the recovery overhead using m-m log to that using checkpointing-only. It can be seen from the data in Fig.4 that the recovery using log is dramaticly faster than that using the checkpointing-only approach. The reason is that processes do not have to wait for the synchronous messages. The proposed approach also incurs a slightly less recovery overhead than the m-MSS-m log approach, and the improvement on the m-MSS-m log may increase with the back off the execution point. One possible explanation is that the transmission latency of messages that are sent out after the failure point in the m-MSS-m log scheme is larger than that in the m-m log scheme. In addition, we find out that the proposed scheme provides a less recovery overhead with a larger message sending rate.

6 Conclusion and Future Work

After discussing the traditional recovery protocols, this paper proposes a message logging protocol for a mobile host to mobile host communication wireless network, which benefits from the decreasing contention and message transmission latency, and then the correctness of algorithms is proofed. The performance of the proposed approach is evaluated with the simulation results, which indicates that our approach provides a better performance in terms of the fail-free overhead and recovery overhead than the traditional approaches. In future, we will study the protocol on unreliable MSSs and compare the performance with the conventional schemes based on a real system.

Acknowledgement The authors would like to thank the anonymous reviewers of this paper for their insightful comments and suggestions.

References:

[1] Pradhan DK, Krishna P, Vaidya NH. Recovery in mobile environments design and trade-off analysis. In: Tohma Y, ed. Proc. of the 26th Int'l Symp. Fault-Tolerant Computing. Sendai: IEEE Press, 1996. 16−25.

[2] Koo R, Touge S. Checkpoinging and rollback-recovery for distributed systems. IEEE Trans. on Software Engineering, 1987,13(1):23−31.

[3] Kim JL, Park T. An efficient algorithm for checkpointing recovery in distributed systems. IEEE Trans. on Parallel and Distributed Systems, 1993,4(8):955960.

[4] Chandy KM, Lamport L. Distributed snapshots: Determining global states of distributed systems. ACM Trans. on Computer Systems, 1985,3(1):63−75.

[5] Ramanathan P, Shin KG. Use of common time base for checkpointing and rollback recovery in a distributed system. IEEE Trans. on Software Engineering, 1993,19(6):571−583.

[6] Elnozahy EN, Johnson DB. The performance of consistent checkpointing. In: Harris C, ed. In: Proc. of the 11th Symp. on Reliable Distributed Systems. Houston: IEEE Press, 1992. 86−95.

[7] Silva LM, Silva JG. Global checkpointing for distributed programs. In: Harris C, ed. Proc. of the 11th Symposium on Reliable Distributed Systems. Houston: IEEE Press, 1992. 155−162.

[8] Prakash R, Singhal M. Low-Cost checkpointing and failure recovery in mobile computing systems. IEEE Trans. on Parallel and Distributed Systems, 1996,7(10):1035−1048.

[9] Manivannan D, Singhal M. Quasi-Synchronous checkpointing: Models, characterization and classification. IEEE Trans. on Parallel and Distributed Systems, 1999,10(7):703−713.

[10] Guohong C, Singhal M. Mutable checkpoints: A new checkpointing aporach for mobile computing systems. IEEE Trans. on Parallel and Distributed Systems, 2001,12(2):157−172.

[11] Wang YM. Maximum and minimum consistent global checkpoints and their applications. In: Sipple RS, ed. Proc. of the 14th Symp. on Reliable Distributed Systems. Bad Neuenahr: IEEE Press, 1995. 86−95.

[12] Randell BL. System structure for software fault tolerance. IEEE Trans. on Software Engineering, 1975,1(2):16−25.

[13] Wang YM, Fuchs WK. Lazy checkpoint coordination for bounding rollback propagation. In: Werner R, ed. Proc. of the 12th Symp. on Reliable Distributed Systems. Princeton: IEEE Press, 1993. 78−85.

[14] Alvisi L, Marzullo K. Message logging: Pessimistic, optimistic, causal, and optimal. IEEE Trans. on Software Engineering, 1998,24(2):145−149.

[15] Elnozahy EN, Zwaenepoe W. Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit. IEEE Trans. on Computers, 1992,41(5):526−531.

[16] Yao B, Ssu KF, Fuchs WK. Message logging in mobile computing. In: Martin DC, ed. Proc. of the 29th Fault-Tolerant Computing Symp. Madison: IEEE Press, 1999. 14−19.

[17] Park T, Yeom HY. An asynchronous recovery scheme based on optimistic message logging for mobile computing systems. In: Werner B, ed. Proc. of the 20th Int'l Conf. on Distributed Computing Systems. Taipei: IEEE Press, 2000. 436−433.

[18] Venkatesan S. Optimistic crash recovery without changing application messages. IEEE Trans. on Parallel and Distributed Systems, 1997,8(3):263−271.

[19] Rao S, Vin HM. The cost of recovery in message logging protocols. In: Palagi L, ed. Proc. of the 17th Symp. on Reliable Distributed Systems. West Lafayette: IEEE Press, 1998. 10−18.

[20] Pei D, Wang DS, Shen MM, Zheng WM. WOB: A novel approach to checkpoint active files. Acta Electronica Sinica, 2000,28(5):9−12 (in Chinese with English abstract).

[21] Li KY, Yang XZ. Improving the performance of a checkpointing scheme with task duplication. Acta Electronica Sinica, 2000,28(5):33−35 (in Chinese with English abstract).

[22] Wei XH, Ju JB. SFT: A consistent checkpointing algorithm with short freezing time. Chinese Journal of Computers, 1999,22(6): 645−650 (in Chinese with English abstract).

[23] Wang DS, Shen MM, Zheng WM, Pei D. A checkpoint-based rollback recovery and processes migration system. Journal of Software, 1999,10(1):69−73 (in Chinese with English abstract).

[24] Lamport,L. Time, clocks, and the ordering of events in distributed systems. Communications of the ACM, 1978,21(7):558−565.

[25] Higaki H, Takizawa M. Checkpointing-Recovery protocol for reliable mobile systems. In: Palagi L, ed. Proc. of the 17th Symp. on Reliable Distributed Systems. West Lafayette: IEEE Press, 1998. 93−99.

附中文参考文献:

[20] 裴丹,汪东升,沈美明,郑纬民.WOB:一种新的文件检查点设置策略.电子学报,2000,28(5):9−12.

[21] 李凯原,杨孝宗.提高用任务重复的检查点方案的性能.电子学报,2000,28(5):33−35.

[22] 魏晓辉,鞠九滨.SFT:一个具有较短冻结时间的一致检查点算法.计算机学报,1999,22(6):645−650.

[23] 汪东升,沈美明,郑纬民,裴丹.一种基于检查点的卷回恢复与进程迁移系统.软件学报,1999,10(1):69−73.

****

第 1 届中国分类技术及应用研讨会(CSCA 2005)

征 文 通 知

2005 年 9 月 23−25 日 北京

CSCA 2005 由中国计算机学会人工智能与模式识别专业委员会主办，由北京交通大学承办。分类是知识处理的基本问题，本次会议旨在推动分类技术研究及相关应用的发展，促进相关科技单位和个人的科技合作和学术交流，以及探讨分类与数据分析技术的研究与应用所面临的挑战性问题及关键性研究课题。会议录用论文将由《计算机研究与发展》（正刊，增刊）正式出版，会议还将评选大会优秀论文和研究生优秀论文。我们诚征有关分类和数据分析领域的最新创新性成果，包括分类和数据分析的原理、方法、算法以及特定领域的实际应用等。

征稿范围（不局限于下述范围）：

分类技术基础理论：监督学习，半监督学习，聚类技术，PLS 路径建模和分类，集成分类技术，多标签分类和 Preference 学习，多事例分类，Multimode 聚类和降维，差异性和聚类结构，分类和聚类算法复杂性……

领域相关的分类和聚类技术：数据密集场景中的分类，文本分类和聚类，Web 页面分类和聚类，时间序列的分类和聚类， 图像与视频检索，计算机视觉中的分类，生物特征识别中的分类……

分类技术应用：银行、金融、保险、市场营销、经济分析，商务智能，知识工程，目标识别，生物信息学、生物统计学，医药和健康科学，信息安全……

投稿要求：(1) 论文应是未发表的研究成果，论文要求中文，采用 word 文件排版，论文请参照《计算机研究与发展》网页“作者须知”中的“最终修改稿要求”（http://crad.ict.ac.cn）书写，论文格式参考 2005 年第 1 期执行。(2) 会议论文采用网上提交方式，在提交论文的同时，必须提交一份投稿声明（从 http://crad.ict.ac.cn 网站下载），作者逐一签字后邮寄或传真到大会会务组，对不提交投稿声明的论文，会议将不予受理。

重要日期：截稿日期 2005-04-25，录用通知日期 2005-05-25，论文提交日期 2005-06-10

来稿请寄：100044 北京交通大学计算机学院 联系人：田凤占

电话：010-51688451，传真：010-51840526，E-mail: fztian@center.njtu.edu.cn