JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 20, 379-390 (2004)

Short Paper

Management of Fault Tolerance Information for Coordinated Checkpointing Protocol without Sympathetic Rollbacks

KWANG SIK CHUNG, YOUNGJUN LEE[*], HEONCHANG YU[**] AND WONGYU LEE[**]
Department of Computer Science
University College London
Gower Street, WC1E 6BT, London
[*]*Department of Computer Education*
Korea National University of Education
Chungbuk, 363-791 Korea
[**]*Department of Computer Science Education*
Korea University
Seoul, 136-701 Korea

This paper presents the condition for an extended global recovery line for coordinated checkpointing protocol and a new garbage collection protocol on checkpoints and message logs in order to avoid the sympathetic rollback caused by lost messages. Since previous works assumed the communication channel does not lose the in-transit messages, those works on garbage collection in coordinated checkpointing protocols delete all the checkpoints except for the last checkpoints on each process. But coordinated checkpointing protocol based on the communication protocol with reliability (TCP) causes in-transit messages to be lost when a failure occurs, and lost messages lead to sympathetic rollbacks of faulty processes or related processes. Thus there is a need for management methods of fault tolerance information that can store and delete the coordinated checkpoint and light message log to avoid sympathetic rollback.

In this paper, we define the extended global recovery line conditions for garbage collection of checkpoints and message logs for lost messages, and present the new garbage collection algorithm within the extended global recovery line. The proposed algorithm uses piggybacked process information on each message so that the additional messages for garbage collection and extended global recovery line are not needed. Since it relies on the piggybacked checkpoint information in communication message, the proposed garbage collection algorithm is called 'the lazy garbage collection algorithm'.

Keywords: coordinated checkpointing protocol, message log, garbage collection, sympathetic rollback, garbage collection

1. INTRODUCTION

With the development of distributed computing systems, one task is partitioned into and run on several processes. The changes of computing environments lead to a

Received June 3, 2002; revised July 15, 2003; accepted August 8, 2003.
Communicated by Jang-Ping Sheu.

higher probability that a failure will occur. Thus fault tolerant techniques have been studied in various forms. There are generally two fault tolerant methods: message logging and checkpointing. The fault tolerance information is saved in volatile memory or stable storage in both of them. The saved fault tolerance information will increase memory consumption and some information become useless (i.e. garage) as time goes on. So a garbage collection is needed. In checkpointing methods, states of processes are stored as fault tolerance information, and useless information for garbage collection as old checkpoints. In message logging methods, the message data logs and deterministic order of message sending and receiving are stored, and message logs are deleted as a garbage [3, 5, 8].

In checkpointing, garbage collection is the process of discarding all fault tolerance information relating to events that occurred before recovery line [2, 5, 7] according to identify consistent recovery line. Since the faulty processes and related processes are rolled back only to the latest checkpoint [4], coordinated checkpointing methods can delete the fault tolerance information before the recovery line.

Coordinated checkpointing methods based on the communication protocol with reliability (TCP) cause lost messages when a failure occurs, and lost messages lead to sympathetic rollbacks of faulty processes or related processes. Therefore we need a new management method for fault tolerance information and order to avoid lost messages and not affect an old consistent recovery line. Rollback recovery of coordinated checkpointing methods needs information on lost messages and garbage collection methods of lost messages.

In this paper we propose the condition for an extended global recovery line for coordinated checkpointing protocol and a new garbage collection protocol based on an extended global recovery line's condition. The paper proceeds as follows. Section 2 presents the system model and the problems of previous coordinated checkpointing protocols on TCP. The motivations are given in section 3. Section 4 proposes conditions of garbage collection of coordinated checkpointing protocol with reliable communication protocol and a garbage collection method of fault tolerance information. Section 5 proves the correctness of our proposed extended global recovery line's condition. Section 6 and section 7 present a comparison, a conclusion and future work.

2. SYSTEM MODEL

Assume that the distributed system consist of N processes. The work unit of the system is denoted by ρ and the message by m during ρ. For each message m that is delivered during ρ, fault tolerance information consists of the identity of the sender process *m.source*, send sequence number *m.ssn* and receive sequence number *m.rsn*. $deliver_{m.dest}(m)$ indicate that a message m is delivered by a process *m.dest* and all states of the system are transmitted [3].

Global states of distributed computing systems consist of all states of related processes and channels [6]. A consistent system state means that if a process state is transmitted by a message receive event, then the message send event should be recorded by the sender process. A consistent global checkpoint for coordinated checkpointing is defined in Definition 1 [6].

Definition 1 A consistent global checkpoint: Let $G\_\ Ckpt = \{C_1, C_2, C_3, \ldots, C_N\}$ be a set of N checkpoints from each process. $G\_\ Ckpt$ is defined as a consistent global checkpoint if, for any message M and any integer $i(1 \leq i \leq N)$, the following condition is satisfied:

$$\exists m,\ receive(M) \in G\_Ckpt \Rightarrow send(M) \in C_i$$ ❑

First, we define the process set that depends on an event in order to define fault tolerance information. *Depend*(m) is the set of processes whose state reflects the delivery of a message m and a sending message m' occurring after m.

Definition 2 *Depend*(m) is defined as follows:

$$Depend\ (m) \overset{def}{=} \left\{ j \in N \mid \begin{array}{c} (j = m.dest \quad \wedge\ j \ \ has \ \ delivered \quad m) \\ \vee\ (\exists m' : (deliver_{m.dest}(m) \rightarrow \ deliver_j(m')) \end{array} \right\}$$

where → denotes the happen-before relation of Lamport [1].

FTI(m), *Fault tolerance information* of message m in rollback recovery is defined in Definition 3.

Definition 3 *FTI*(m) that is fault tolerance information of message m is defined as follows:

i) $FTI(m) \overset{def}{=} \{content_m, m.dest, m.rsn, m.ssn\} \cup \{Depend(m)\}$

ii) *FTI*(m) is saved as checkpoints or message logs. ❑

3. MOTIVATIONS

Messages that are sent but not yet received are called in-transit messages. According to Definition 1, even though a failure occurs, the system always recovers to a consistent state if a system has a consistent global recovery line. It is possible by assumptions that a communication channel is reliable and an in-transit message is always consistent.

Previous works exclude in-transit messages from consistent global checkpoints and assume they are part of a communication channel. Thus in-transit messages don't make the global system state inconsistent. Depending on the assumption of reliablility of communication channels, rollback-recovery protocols may have to guarantee the delivery of in-transit messages when a failure occurs [2].

Rollback recovery protocol may be implemented based on a reliable communication protocol or based on unreliable communication channel. In the case of a rollback recovery protocol is implemented on top of an unreliable communication channel, the rollback recovery protocol does not assume reliable communications.

The reliable communication protocols guarantee the delivery of in-transit messages in failure free executions. But, the protocols can not guarantee the delivery of in-transit messages when a failure occurs. For example, if an in-transit message is lost due to the failure of receiver process, the convenient communication protocols would inform the sender process of the loss of the in-transit message through the time-out protocol. But rollback-recovery protocols should guarantee the recovery of the faulty receiver process and support time-out protocol. Thus rollback-recovery protocol should resend the lost message to the process after the receiver process recovers.

If the system model assumes the unreliable communication channel, rollback-recovery protocol doesn't have to manipulate the in-transit messages. Lost in-transit messages due to process failure are not distinguished from those due to failure in an unreliable communication channels. The loss of in-transit messages can occur during failure free execution [2].

The rollback-recovery protocol based on reliable communication protocol (e.g., TCP) would be implemented without the delivery of in-transit messages on a failure time. If the system model assumes the reliable communication protocol, lost messages have to be manipulated in the recovery protocol.

4. CONDITION FOR EXTENDED GLOBAL RECOVERY LINE AND GARBAGE COLLECTION

4.1 Condition for Extended Global Recovery Line

Sender processes have to maintain message logs or checkpoints of in-transit messages. In order to log messages, we use the sender-based message logging protocol. This protocol can save message logs in volatile memory since it doesn't rollback before the latest checkpoint. Our extended recovery line is proposed as bellow.

Condition 1 When the rollback-recovery protocol is based on the coordinated checkpointing and reliable communication protocol, and process P_i sends a message to P_j, the condition for the extended recovery line of fault tolerance information $FTI(m)$ is

i) $receive_i(m) \in G\_Chpt \Rightarrow send_i(m) \in chpt_i^k$ and,

ii) $\exists chpt_j^{P\text{-}1}$, $chpt_j^{P\text{-}1} \rightarrow receive_j(m)$ and,

iii) $\exists chpt_j^P$, $receive_j(m) \rightarrow chpt_j$. ❑

If Condition 1 is satisfied, our garbage collection method of fault tolerance information that is related to in-transit messages or lost messages is valid. When all three parts of Condition 1 are satisfied, $garbage_i(FTI(m))$ is valid. $chpt_j^P$ is the checkpoint that is taken by process P_j after receiving the event of m, and $garbage_i(FTI(m))$ is the garbage collection event of by $FTI(m)$. In Definition 2, $FTI(m)$ is assumed to be stored as a message log. According to garbage collection condition of $FTI(m)$, when the receiver process takes a checkpoint after receiving a message, $FTI(m)$ can be deleted as a garbage. For garbage collection, we define the fault tolerance information.

In Fig. 1, before time t_1, processes P_2, and P_4 send messages m_2, m_3 to P_3, respectively. At time t_1, processes P_2 and P_4 maintain the message log of m_2 and m_3. Although process P_3 fails between time t_1 and time t_2, messages m_2 and m_3 can be restored.

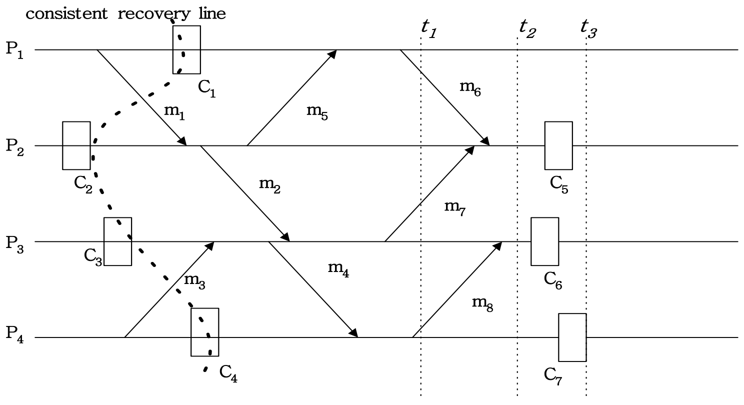

Fig. 1. Process information relating to lost message.

At time t_3, process P_3 doesn't need to rollback to C_3. The message log of messages m_2 and m_3, is not needed to be saved in volatile storage and can be deleted as garbage. According to Condition 2, message logs of a process can be deleted as garbage.

The garbage collection condition relating to lost message is as follows.

Condition 2 Garbage collection condition of information relating to lost messages:

When a process P_i sends a message to process P_j, P_i stores the checkpoint interval (CI_j) of P_j relating to lost message $Lost\_MSG\_Info_i^j[CI_j]$. The P_i can delete the message logs with smaller checkpoint intervals than the checkpoint interval piggybacked with messages from P_j. ❑

According to the extended recovery line of Condition 1, garbage collection of message *m*'s *FTI*(*m*) is restricted within narrow limits. And, Condition 2 defines the garbage collection limits of Condition 1. The limited garbage collection without sympathetic rollback can be decided by comparing checkpoint intervals ($CI_{\mathrm{Pr}\,ocessID}$) of the receiver process of message *m*. By comparing $CI_{\mathrm{Re}\,ceiver\,\mathrm{Pr}\,ocess}$ in sender process and $CI_{\mathrm{Re}\,ceiver\,\mathrm{Pr}\,ocess}$ in the receiving message, we can decide whether the receiver process is included in extended recovery line or not. If the receiver process is included in the extended recovery line, then message *m*'s *FTI*(*m*) can be deleted.

4.2 Garbage Collection Protocol

This section describes the garbage collection algorithm for message logs using $Lost\_MSG\_Info_i^j[CI_j]$ defined in Condition 2.

When a process P_i sends a message *m* to a process P_j, P_i saves message logs of *m* in volatile storage. If coordination of checkpoints is initiated and P_i receives a message piggybacked with checkpoint interval information, message logs of *m* can be deleted after comparing checkpoint intervals. If an initiator process in coordinated checkpointing is the sender of message *m*, then the sender can delete message logs with a smaller checkpoint interval than the checkpoint interval of the sender. Such garbage collection of the message logs doesn't make the sympathetic rollback.

Fig. 2 shows the garbage collection algorithm for message logs using fault tolerance information.

```
Pr ocess $P_i$ sends a message $M$ to $P_j$ and process $P_i$ maintains a message log in
volatile storage.

i) Pr ocess $P_i$ receivesmessageN from process $P_k$
    pivot_CI = each processes' Checkpoint_Intervals in N from $P_k$
    while ($P_k \in P_i$'s related process list) {
        if (pivot_CI > $Lost\_MSG\_Info_i^j$ [$CI_j$])
            garbage$_i$ (FTI (M))
        }
ii) Pr ocess $P_i$ receives a coordinated checkpo int request from another process
    or Pr ocess $P_i$ initiates the coordinated checkpointing procedure
    pivot_process = process id related with coordinated checkpoints
    if (the coordinated checkpo int completed $\wedge$ process $P_i \in$ pivot_process)
    garbage$_i$ (FTI(M))
```

Fig. 2. Garbage collection algorithm for message logs.

4.3 Example of Management of Fault Tolerance Information

Fig. 3 illustrates the management of fault tolerance information with the Garbage collection algorithm (Fig. 2) and Condition 2. If the message satisfies Condition 2, the message can be deleted from the message log and Condition 2 is checked in the Garbage collection algorithm.

For example, at time t_1 in Fig. 3, process P_4 has to keep the message logs of messages 8 and 12, since if the other processes have a fault, then the messages 8 and 12 will be used for fault recovery. But, P_4 can delete the message log of message 11, since for message 11 CI_{P_1} is greater than CI_{P_4} (Condition 2). At time t_2, message 16 is added to the message log of P_4, since message 16 will make a sympathetic rollback of P_3 when a fault occurs at P_4. At time t_3 message 18 is added to the message log of P_4. If the processes coordinate the global checkpoints or P_4 receives a message with a checkpoint interval, P_4 may delete some message logs as garbage using the Garbage collection algorithm (Fig. 2). In this case the messages would not make a sympathetic rollback. Further, Condition 2 will validate it as proved in section 5.

At time t_1, message 11 can be deleted from the message logs, since P_5 coordinates the checkpoint with P_1 and P_4 and message 11 is included in recovery line 2. At time t_2, P_1 can delete the message log of message 20, since P_2 is involved in recovery line 3. At time t_3, message 14, 15, 16 and 18 can be deleted because P_2, P_3, P_4 and P_5 are involved in recovery line 4. At time t_4, no message can be deleted as garbage, since any message can not satisfy Condition 2.

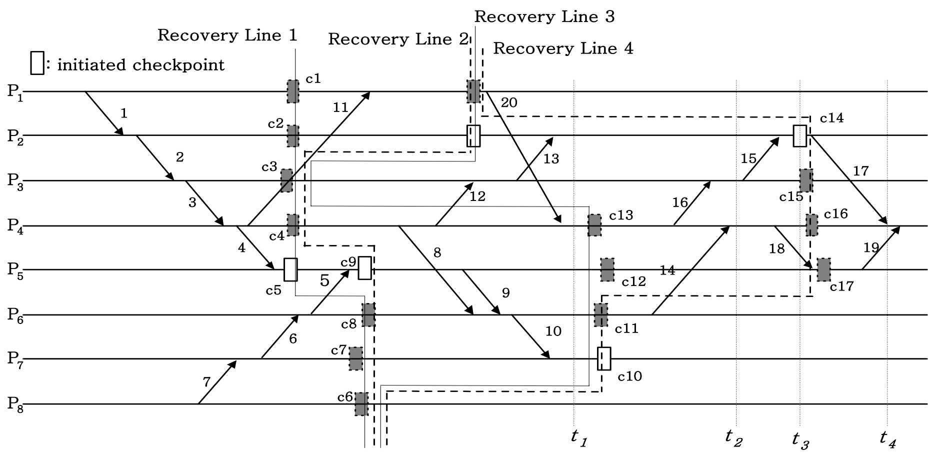

Fig. 3. Example of garbage collection using fault tolerance information.

5. PROOF OF CORRECTNESS

A condition for garbage collection of fault tolerance information is that a process should not make an orphan message when a failure occurs. The conditions for garbage collection are valid only if the system keeps consistent global states after rollback recovery is applied to recover from failures.

Garbage collection of fault tolerance information requires the detection of garbage information and keeping valid states. Theorem 1 and Theorem 2 show that the proposed conditions for garbage collection cause the system keep its consistent global states.

Theorem 1 If we assume that the reliable communication and synchronous checkpointing methods are supported, we would have to maintain not only the latest checkpoints, but also fault tolerance information *FTI*(*m*) in order to avoid sympathetic rollbacks occurring in in-transit messages.

Proof: If coordinated checkpointing protocols are assumed with reliable communication protocol such as TCP, and since each process has a checkpoint that is coordinated with other processes, each process would not be rolled back to the time before the latest checkpoints. Although in-transit messages at the time of failure would not make the whole system inconsistent, a recovery protocol should replay the messages. That is, because in-transit messages at the consistent recovery line would become lost during the recovery procedure and lost messages would make the whole system to be inconsistent.

In order to prevent sympathetic rollback due to lost messages, a recovery protocol needs the other fault tolerance information.

In Fig. 4, due to a failure occurring at time *t*, the process P_2 has to roll back to checkpoint C_3. Process P_2 that is rolled back to C_3 needs to receive the message m_1. Thus the process P_1 has to send m_1 again. If process P_1 does not maintain the message log of m_1, process P_1 has to roll back to C_1. In order to send m_1 again, process P_1 has to replay the processing from C_1. The sender processes have to maintain the fault tolerance information *FTI*(*m*) relating to in-transit messages as well as the latest checkpoint. ❑

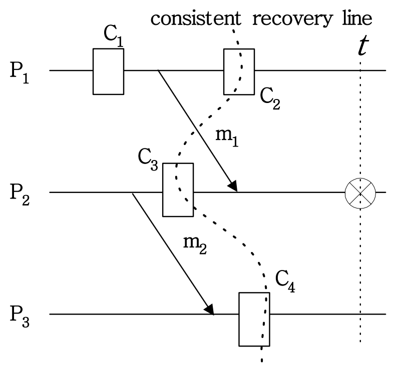

Fig. 4. *FTI*(*m*) for lost message.

Theorem 2 Assume that the reliable communication and synchronous checkpointing methods are supported and a process sends a message to:

(i) $receive_i(m) \in G\_Chpt \Rightarrow send_i(m) \in chpt_i^k$ and,

(ii) $\exists chpt_j^{P\text{-}1}$, $chpt_j^{P\text{-}1} \rightarrow receive_j(m)$ and,

(iii) $\exists chpt_j^P$, $receive_j(m) \rightarrow chpt_j$.

if (i), (ii), and (iii) are satisfied, *garbage*(*FTI*(*m*)) is valid.

Proof: Consider condition (i), $receive_i(m) \in G\_Chpt \Rightarrow send_i(m) \in chpt_i^k$. Messages m_1 and m_2 in Fig. 5 satisfy $send_i(m) \in chpt_i^k$. If we consider conditions (ii) and (iii) with messages m_1 and m_2, there are two possible cases.

(a) In the case of message m_1, at checkpoint C_4 of process, previous checkpoint $chpt_j^{P\text{-}1}$ that satisfies condition (ii) $\exists chpt_j^{P\text{-}1}$, $chpt_j^{P\text{-}1} \rightarrow receive_j(m)$ is C_3. And C_4 satisfies condition (iii). Therefore, message m_1 can be deleted since it would not be a lost message in case of failure.

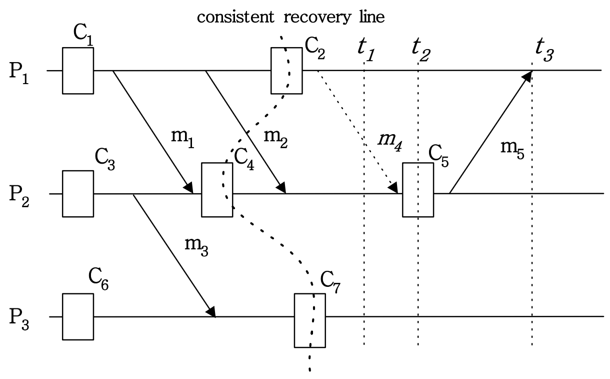

Fig. 5. Garbage collection of FTI(m) for lost message.

We assume that a failure occurs at time t_1. At time t_1, fault tolerance information of lost message m_1 already was deleted. But, the failure makes P_2 roll back to C_4 and message m_1 will not become a lost message. Thus fault tolerance information of message m_1 can be deleted.

(b) In the case of message m_2, previous checkpoint $chpt_j^{P\text{-}1}$ that satisfies condition (ii), $\exists chpt_j^{P\text{-}1}$, $chpt_j^{P\text{-}1} \rightarrow receive_j(m)$ is C_4. If checkpoint C_5 is taken, C_5 will satisfy condition (iii). Thus fault tolerance information of m_2 can be deleted after C_5.

Next, we can decide the time to delete the fault tolerance information of message m_2. If we assume that a failure occurs at time t_1, process P_2 would roll back to checkpoint C_3. Process P_2 would replay and message m_2 would become lost. Thus fault tolerance information for P_1 would be needed. Process P_1 can not delete $FTI(m)$. At time t_1, checkpoint C_5 of process P_2 satisfies condition (iii). If a failure occurs at time t_2, message m_2 would not become lost and fault tolerance information of message can be deleted. But, since process does not know that C_3 was taken, P_1 can not delete the fault tolerance information. At time, process receiving message m_5, knows that C_5 was taken. Thus fault tolerance information for message m_2 can be deleted.

At times t_2 and t_3, garbage collection for fault tolerance information can be performed since there will not be any rollbacks. However, the time of garbage collection for fault tolerance information for P_1 is time t_3, because it receives a message from P_2 at time t_3. ❑

In case (b) of Theorem 2, the time of garbage collection of fault tolerance information for message m_5 is time t_3. But, if C_5 of process P_2 needs to coordinate with process P_1, then process P_1 will take a coordinated checkpoint with process P_2 and know that the checkpoint interval of process P_2 has increased. If at time t_2 conditions (i), (ii), and (iii) will be satisfied, fault tolerance information of message m_2 can be deleted.

6. COMPARISON

The main points of our work based on coordinated checkpoint methods [8-12] are the detection of consistent recovery lines for checkpoints and garbage collection according to globally consistent recovery lines. The garbage collection of coordinated checkpoint methods is based on the fact that all processes of the system will not roll back after the global recovery lines. Thus all checkpoints before the globally consistent recovery line can be deleted and are based on Theorem 1. If coordinated checkpointing methods were implemented based on the communication protocol with reliability (TCP), then it could cause lost messages when a failure occurs, and lost messages lead to sympathetic rollbacks of faulty or related processes. In order to avoid these sympathetic rollbacks, we delay deleting the last checkpoints before the globally consistent recovery lines.

In previous works, the recovery interval is the execution time between a checkpoint and the other checkpoint of one process. The fault tolerance information can be deleted according to Definition 1. We proposed the extended recovery lines and constraint on garbage collection of the fault tolerance information. After the extended recovery line is completed, the fault tolerance information can be deleted. Also, we proved that lazy garbage collection of fault tolerance information based on Condition 1 does not result in sympathetic rollback by Theorem 1 and Theorem 2. Also we prove that the additional messages are not needed for the proposed lazy garbage collection.

7. CONCLUSION AND FUTURE WORKS

We proposed conditions for an extended global recovery line for coordinated checkpointing protocol and a garbage collection protocol for message logs saved in volatile storage and checkpoints in stable storage to avoid the sympathetic rollback caused by lost messages. If the recovery protocol is based on a reliable communication protocol such as TCP, using only the latest checkpoint for rollback recovery causes processes to roll back sympathetically.

In order to avoid sympathetic rollback, we proved that other checkpoints and message logs in addition to the latest checkpoint have to be maintained. We defined the conditions for the global recovery line's condition for coordinated checkpointing protocol and proposed the garbage collection of checkpoints and message logs to avoid sympathetic rollback, and also proposed a garbage collection algorithm.

Since the proposed algorithm uses computational messages piggybacked with fault tolerance information on in-transit messages, our garbage collection algorithm does not generate additional messages for garbage collection. Fault tolerance information is piggybacked with computational messages and includes only the checkpoint interval of a sender process.

If a process does not receive any message from other processes or is not involved in a coordinated checkpoint group, it can not delete the fault tolerance information. But, "lazy garbage collection" does not make the whole system inconsistent. Our future effort is to solve the problem of "lazy garbage collection".

REFERENCES

1. K. M. Chandy and L. Lamport, "Distributed snapshots: determining global states of distributed systems," *ACM Symposium on Principles of Database System*, Vol. 3, 1985, pp. 63-75.
2. M. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson, "A survey of roll-back-recovery protocols in message-passing systems," Technical Report CMU-CS-96-181, Dept. of Computer Science, Carnegie Mellon University, 1996
3. D. B. Johnson and W. Zwaenpoel, "Sender-based message logging," in *Proceedings of the Seventeenth International Symposium on Fault-Tolerant Computing*, 1987, pp. 14-19.
4. R. Koo and S. Toueg, "Checkpoint and rollback-recovery for distributed systems," *IEEE Transactions on Software Engineering*, Vol. 13, 1987, pp. 23-31.
5. Y. Liu and J. Chen, "On thorough garbage collection in distributed systems," in *Proceedings of 3rd IEEE Symposium on Computers and Communications*, 1998, pp. 576-581.
6. D. Manivannan and M. Singhal, "A low-overhead recovery technique using quasi-synchronous checkpointing," in *Proceedings of the 16th International Conference on Distributed Computing Systems* (*ICDCS*), 1996, pp. 100-107.

7. M. V. Sreenivas and S. Bhalla, "Garbage collection in message passing distributed systems," *First Aizu International Symposium on Parallel Algorithms/Architecture Synthesis*, 1995, pp. 213-218.
8. J. Xu, R. H. B. Netzer, and M. Mackey, "Sender-based message logging for reducing rollback propagation," *Seventh IEEE Symposium on Parallel and Distributed Processing*, 1995, pp. 602-609.
9. E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The performance of consistent checkpointing," in *Proceedings of IEEE Symposium on Reliable Distributed Systems*, 1992, pp. 39-47.
10. P. Ramanathan and K. G. Shin, "Use of common time base for checkpointing and rollback recovery in a distributed system," *IEEE Transactions on Software Engineering*, Vol. 9, 1993, pp. 571-583.
11. Z. Tong, R. Y. Kim, and W. T. Tsai, "Rollback recovery in distributed systems using loosely synchronized clocks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, 1992, pp. 246-251.
12. L. M. Silva and J. G. Silva, "Global checkpointing for distributed program," in *Proceedings of IEEE Symposium on Reliable Systems*, 1992, pp. 155-162.
13. Y. Manabe, "A distributed consistent global checkpoint algorithm for distributed mobile system," in *Proceedings of Eighth International Conference on Parallel and Distributed Systems* (*ICPADS 2001*), 2001, pp. 125-132.

Kwang Sik Chung received the B.S. degree (1992), the M.S. degree (1995), and the Ph.D. degree (2000) in Computer Science and Engineering from Korea University. He is currently a senior consultant at Samsung SDS Ltd. And from September 2002 to November 2003, he is also a research fellow of Department of Computer Science at University College London. His research interests include distributed systems, fault tolerant systems, grid computing systems.

YoungJun Lee received a B.S. degree (1989), the MS degree (1991), and the PhD degree (1994) in Computer Science from University of Minnesota, MN, USA, in 1994. He is a professor of Graduate School and Department of Computer Education at Korea National University of Education. His research interests include distributed systems, and advanced network and intelligent systems.

HeonChang Yu received the B.S. degree (1989), the M.S. degree (1991), and the Ph.D. degree (1994) in Computer Science and Engineering from Korea University. He is currently an associate professor of Computer Science Education at Korea University in Korea since 1998. And from February 2004 to January 2005, he is also a visiting professor of Computer Science at Georgia Institute of Technology. He was an assistant professor of Computer Engineering at Seokyeong University in Korea. He is a vice president at the Korea Association of Computer Education and a member of the ACM. His research interests include distributed systems, fault tolerant systems, grid computing systems, and agent systems.

WonGyu Lee received the B.S. degree (1985) in English Language and Literature from Korea University, the M.S. degree (1989) and the Ph.D. degree (1993) in Information Sciences and Electronics from University of Tsukuba in Japan. He is currently an associate professor of Computer Science Education at Korea University in Korea since 1996. He was a principal researcher of Cultural Information at Korea Culture & Arts Foundation. He is a president at the Korean Association of Computer Education and a member of the ACM, IEEE. His research interests include database systems, information models, semantic structures, information retrieval, and computer science education.