A Distributed Consistent Global Checkpoint Algorithm for Distributed Mobile Systems

Yoshifumi Manabe
NTT Communication Science Laboratories
2-4, Hikari-dai, Seika-cho, Kyoto, 619-0237 Japan
manabe@cslab.kecl.ntt.co.jp

Abstract

A distributed coordinated checkpointing algorithm for distributed mobile systems is presented. A consistent global checkpoint is a set of states in which no message is recorded as received in one process and as not yet sent in another process. It is used for rollback when process failure occurs. A consistent global checkpoint must be obtained for any checkpoint initiation by any process. This paper shows a checkpoint algorithm in which the amount of information piggybacked on program messages does not depend on the number of mobile processes. The number of checkpoints is minimized under two assumptions: (1) one consistent global checkpoint is taken for concurrent checkpoint initiations and (2) a checkpoint is initiated at each handoff by mobile processes. This algorithm is thus optimal among the generalizations of Chandy and Lamport's distributed snapshot algorithm under the latter assumption.

1. Introduction

Recent wireless-LAN and personal-computer technology has made mobile computing realizable. This paper considers distributed mobile systems consisting of mobile hosts (MHs) and static hosts. Static hosts are connected by a wired LAN and their physical locations do not change. Some static hosts can be mobile support stations (MSSs). An MSS can communicate with MHs by a wireless medium. For simplicity of discussion, this paper assumes that every static host is an MSS. An MSS or an MH is called a process. When an MH moves, its MSS can change. This is called a handoff.

Algorithms for distributed mobile systems must consider the following issues [2].

(M1) The capability of MHs is limited compared to static hosts. Thus, the amount of computation in MH must be minimized.

(M2) The bandwidth of a wireless LAN is lower than that of a wired LAN. Thus, the communication overhead of the wireless medium must be minimized.

(M3) The number of MHs may not be known in advance and it may be larger than the number of static hosts. Thus, the overhead of computation and communication must not increase with the number of MHs.

(M4) Handoffs occur during execution and the algorithm must be able to handle the effects of handoffs.

Causal ordering protocols have been presented [2][9] as a basic protocols for distributed mobile systems. This paper presents a checkpointing protocol for distributed mobile systems. A consistent global checkpoint is a set of states in which no message is recorded as received in one process and as not yet sent in another process [8]. When there is process failure, execution can be continued from the set of rolled-back states if every process rolls back to each state in a consistent global checkpoint and the messages that have been sent and not received are restored. The message restoration method is similar to Strom and Yemini's [12] and the details are discussed in [7]. This paper thus focuses on how to obtain a consistent global checkpoint.

It is assumed that checkpointing might be initiated at any time in any process. When a process initiates checkpointing, it takes its checkpoint and notifies the other processes about the initiation. When each of the other processes receives this information, it might have to take its checkpoint in order to obtain a consistent global checkpoint which contains the initiation. Throughout this paper, the checkpoint taken by the initiator is called an initiation. The other checkpoints are called additional checkpoints. The number of additional checkpoints must be minimized to reduce the overhead of checkpointing.

Many checkpointing algorithms for distributed systems have been reported. Chandy and Lamport's distributed snapshot algorithm [3] has been modified for distributed mobile

0-7695-1153-8/01 $10.00 © 2001 IEEE

systems [11]. Though the algorithm obtains one consistent global checkpoint for concurrent initiations, it deals with each non-consistent initiation independently. Thus, the number of additional checkpoints is not minimized. An index-based algorithm [1] does not minimize the number of additional checkpoints either. An algorithm that minimizes the number of additional checkpoints in distributed systems under Chandy and Lamport's assumption that one consistent global checkpoint is obtained for concurrent initiations has been shown independently [4][6]. However, it uses an array of integers whose size is the number of processes. Thus, the size depends on the number of MHs and property (M3) is not satisfied.

In this paper, the algorithm in [4][6] is modified for distributed mobile systems without using large size variables with one additional assumption. The additional assumption is that an MH takes a checkpoint whenever a handoff occurs. This checkpoint is called a handoff checkpoint. This assumption is introduced in [1] since the possibility of failure might be high at handoff compared to the time of normal operations. This paper assumes that a handoff checkpoint is an additional checkpoint. However, this paper's algorithm can be easily modified for the case when a handoff checkpoint is an initiation; that is, when one consistent global checkpoint that includes each handoff checkpoint must be obtained. If every handoff checkpoint is an initiation, the algorithm minimizes the number of additional checkpoints. Thus, it is optimal among the generalizations of Chandy and Lamport's distributed snapshot algorithm under the assumption that every handoff checkpoint is an initiation.

2. Consistent global checkpoint

A distributed mobile system is modeled by a finite set of mobile support stations $S = \{s_1, s_2, \ldots, s_n\}$ and a set of mobile hosts $H = \{h_1, h_2, \ldots, h_\ell\}$. An element in $S \cup H$ is called a process. The MSSs are interconnected by point-to-point static channels. Each MH is connected to at most one MSS at the same time. Let H_i be the set of MHs currently connected to MSS s_i. The channel between MH and MSS is a wireless channel (Fig. 1). Channels are assumed to be error-free and have infinite capacity. The communication is asynchronous; that is, the delay experienced by a message is unbounded but finite. Channels might not be FIFO (First-In, First-Out).

Every communication between an MH and the other processes is done via the MSS to which it is currently connected. For example, when h sends message m to process $s(\neq s_i)$ where s_i is h's current MSS, h sends a pair (s, m) to s_i and s_i forwards it to s. In the rest of paper, the pair (s, m) is considered as a message from s to s_i. A similar assumption is given in receiving by an MH.

A process $p \in S \cup H$'s execution includes the following

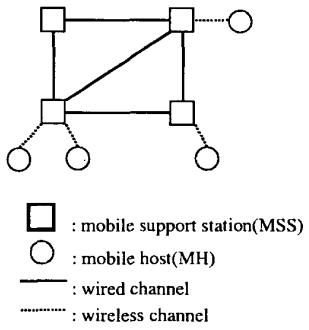

Figure 1. Distributed mobile system.

events called handoff events.

1. Connect: MH h executes *connect* in order to set up a wireless communication channel between MSS s_i.
2. Accept: MSS s_i executes *accept* in response to *connect*[1]. h is added to H_i.
3. Disconnect: MH h executes *disconnect* in order to disconnect the channel between the current MSS s_i.
4. Remove: MSS s_i executes *remove* in response to *disconnect*. h is removed from H_i.
5. Handoff checkpoint: MH h takes a checkpoint between two events, *disconnect* to its old MSS and *connect* to new MSS.

It is assumed that MH can send some information related to handoff to its MSS by *connect*. The MSS can receive it by *accept*. It is also assumed that MH can send some information related to handoff by *disconnect* and MSS can receive it by *remove*. Thus, a *connect*(*disconnect*) and *accept*(*remove*) event can be considered as a special kind of *send* and *receive* event. In the rest of the paper, $s(m)$ means send event, connect event, or disconnect event whose content is m. Similarly, $r(m)$ means receive event, accept event, or remove event. Since the channel may be non-FIFO, some messages between MH and its MSS might not have been received at *disconnect* and *remove*. These messages could be lost. This makes no problem in checkpointing, since (1) from MH to MSS, *disconnect* message can also piggyback information necessary for checkpointing, and (2) from MSS to MH, the lost messages make no need to take a checkpoint in MH.

Checkpoint initiation events are executed independently by each process in $S \cup H$. System execution E is the set of

[1] In [11], *accept* occurs before *connect*. This is a minor change since any communication between MH and MSS can be done after these two events anyway.

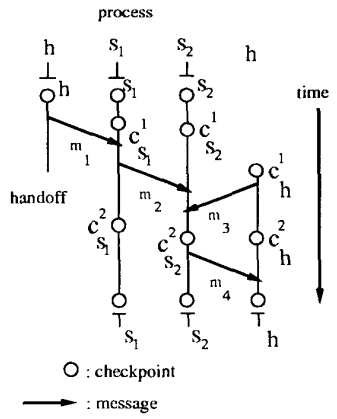

Figure 2. System execution.

all processes' executions. Checkpointing algorithm A takes additional checkpoints so that there is a consistent global checkpoint which includes each initiation. Process p's execution with checkpointing algorithm A is p's execution interleaved with the additional checkpoints taken by A in p. System execution with A, $E(A)$, is the set of all processes' executions with A. Since a handoff checkpoint is not an initiation, there might be no consistent global checkpoint that includes a given handoff checkpoint. If such a consistent global checkpoint is needed, it is achieved by executing the procedure for initiations at the handoff checkpoint.

In the rest of the paper, a sequence number is assigned to any (initiation and additional) checkpoint in a process. Let c_p^x be p's x-th checkpoint.

Fig. 2 shows an example of system execution. Initially, MH h is connected to s_1. h then sends disconnect $s(m_1)$ and connect $s(m_3)$ to s_2. c_h^1 is a handoff checkpoint.

The "happened before($\rightarrow$)" relation between the events in $E(A)$ is defined as follows [5].

Definition 1 *$e \rightarrow e'$ if and only if*

(1) e and e' are executed in the same process and e is not executed after e'.

(2) e is the send event $s(m)$ and e' is the receive event $r(m)$ of the same message m.

(3) $e \rightarrow e''$ and $e'' \rightarrow e'$ for event e''. ■

Note again that $s(m)$ can be connect or disconnect in (2).

When e and e' are executed in different processes and $e \rightarrow e'$, there is a sequence of events $e, s(m_1), r(m_1), s(m_2), \ldots, s(m_k), r(m_k), e'$ in which $e \rightarrow s(m_1)$, $r(m_i) \rightarrow s(m_{i+1})(i = 1, \ldots, k-1)$, $r(m_k) \rightarrow e'$ and these pairs of events are executed in the same process. This sequence is called a causal sequence from e to e'.

Two special events, $\perp_p$ and $\top_p$, are defined for process p. $\perp_p$ is an imaginary event which is p's initial state. For any event e in p, $\perp_p \rightarrow e$. $\top_p$ is p's current event if p is not terminated. If p is terminated, $\top_p$ is an imaginary event which is p's terminal state. For any event e in p, $e \rightarrow \top_p$. This paper considers $\perp_p$ and $\top_p$ as checkpoints in E.

Definition 2 *A pair of checkpoints (c, c') is consistent if and only if $c \not\rightarrow c'$ and $c' \not\rightarrow c$.*

A global checkpoint $(c_1, c_2, \ldots, c_N)$ is N-tuple of checkpoints which consists of one checkpoint in each process, where N is the number of processes. A global checkpoint is consistent if and only if all distinct pairs of checkpoints are consistent. ■

In Fig. 2, $(c_{s_1}^1, c_{s_2}^1, c_h^2)$ is consistent, but $(c_{s_1}^1, c_{s_2}^2, c_h^2)$ is not consistent because $c_{s_1}^1 \rightarrow c_{s_2}^2$.

A consistent global checkpoint for process p's checkpoint initiation c_p in $E(A)$ is denoted as $gc(c_p, E(A))$. q's checkpoint in $gc(c_p, E(A))$ is denoted as $gc(c_p, E(A), q)$. $E(A)$ is omitted if it is obvious.

Checkpointing algorithms use either special messages called markers [3] or are communication-induced algorithms [4][6][10], in which all information for checkpointing is piggybacked on massages in E. This paper discusses the latter type, since markers are not effective in non-FIFO channels [6]. Note that checkpointing algorithms can piggyback information also on connect/disconnect messages and the information can be received by accept/remove events.

3. The checkpointing algorithm

If the algorithm in [4][6] is used for the distributed mobile system, the amount of information piggybacked on each message and maintained by each process is $O(|S| + |H|)$ integers. This size can be reduced by using the fact that MH sends every message via MSS. In the present algorithm, the amount of information piggybacked on each message between MSSs is $O(|S|)$ integers and the one between MSS and MH is $O(1)$ integers. MH maintains $O(1)$ integers and MSS maintains $O(|H_i| \cdot |S|)$ integers. Thus, the requirements for mobile systems, (M1), (M2), and (M3) are achieved for MH because MH maintains, sends, and receives $O(1)$ integers during execution. Because each MSS has a limited number of wireless channels, $|H_i|$ is limited for MSS s_i and (M3) is achieved for MSS.

The basic algorithm in [6] consists of two rules, (Rule 1) and (Rule 2).

(Rule 1) A global checkpoint number (GCN) is assigned to each initiation. The initiations with the same GCN are

concurrent. One consistent global checkpoint is obtained for each GCN.

(Rule 2) An additional checkpoint is taken when a consistent global checkpoint might not be obtained for a GCN.

(Rule 1) reduces the number of additional checkpoints compared to the case when a consistent global checkpoint is obtained for each initiation independently. The GCN assignment is exactly the same as Lamport's logical clock [5]. In Fig. 2, GCN 1 is assigned to $c^1_{s_1}$, $c^1_{s_2}$, and c^2_h and GCN 2 is assigned to $c^2_{s_1}$ and $c^2_{s_2}$ when all of these checkpoints are initiations.

One implementation of (Rule 1) for distributed mobile systems is as follows. MSS s_i has an integer vector gcn_i, whose size is $|S|$. s_i also has an integer $hgcn_i(h)$ for each MH $h \in H_i$. $gcn_i(j)$ and $hgcn_i(h)$ maintains knowledge about GCN in each process. $gcn_i(j) = y$ means that s_i knows that s_j knows the maximum GCN is y. $hgcn_i(h) = y$ means that s_i knows that h knows the maximum GCN is y. $gcn_i(i)$ is the current GCN s_i knows. MH h has an integer gcn, which is the current GCN h knows. In Fig. 2, $gcn_2(1) = 1$, $gcn_2(2) = 2$, and $hgcn_1(h) = 0$ at $c^2_{s_2}$ in process s_2. $gcn = 1$ at c^2_h in process h.

GCN is assigned using these variables as follows:

(Rule 1 in MSS: GCN assignment rule for MSS s_i)

(1) Initially, set $gcn_i(j) := 0$ for all j and $hgcn_i(h) := 0$ for all $h \in H_i$.

(2) When initiation $c^x_{s_i}$ occurs, increment $gcn_i(i)$ and assign it as the GCN for $c^x_{s_i}$.

(3) When s_i sends a message m to a MSS, the current value of gcn_i is piggybacked on m.

(4) When s_i sends a message m to MH $h \in H_i$, the current value of $gcn_i(i)$ is piggybacked on m.

(5) When s_i receives a message m from MSS s_j, let the value of gcn_j on m be $mgcn$.
Set $gcn_i(k) := max(gcn_i(k), mgcn(k))$ for each k and then set $gcn_i(i) := max(gcn_i(i), mgcn(j))$.

(6) When s_i receives a message m from MH h, let the value of gcn on m be $mgcn$.
Set $hgcn_i(h) := max(hgcn_i(h), mgcn)$ and then set $gcn_i(i) := max(gcn_i(i), mgcn)$.

(7) When MH h is connected to s_i, s_i receives the value of gcn_j at the last send event to h in h's previous MSS s_j[2]. Let the received value be $hlgcn$. Set $gcn_i(k) := max(gcn_i(k), hlgcn(k))$ for each k and then set $gcn_i(i) := max(gcn_i(i), hlgcn(j))$.
Next let the value of $hgcn$ on the request message be $mgcn$. Set $hgcn_i(h) := max(gcn_i(h), mgcn)$ and then set $gcn_i(i) := max(gcn_i(i), mgcn)$.

[2]The procedure for sending this information is shown later.

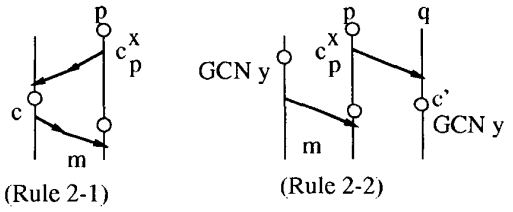

Figure 3. Rules for taking an additional checkpoint.

(8) When MH h is disconnected from s_i, let the value of gcn on the request message be $mgcn$.
Set $hgcn_i(h) := max(hgcn_i(h), mgcn)$ and then set $gcn_i(i) := max(gcn_i(i), mgcn)$.

(Rule 1 in MH: GCN assignment rule for MH h)

(1) Initially, set $gcn := 0$.

(2) When initiation c^x_h occurs, increment gcn and assign it as the GCN for c^x_h.

(3) When h sends a message or a connect/disconnect request message m to MSS, current value of gcn is piggybacked on m.

(4) When h receives a message m from MSS s_j, let the value of $gcn_j(j)$ on m be $mgcn$.
Set $gcn := max(gcn, mgcn)$.

(Rule 2) in [6] is the rule for taking an additional checkpoint. It consists of two subrules. Consider the case when process p receives information about a new GCN from another process at event $r(m)$. (Note again that $r(m)$ can be a receive/accept/remove event.) This is the case when $gcn_i(i) < mgcn(j)$(or $mhgcn$) is satisfied (before updating $gcn_i(i)$) in (5)(6)(7)(8) for MSS s_i and $gcn < mgcn$ is satisfied in (4) for MH h. Let p's newest checkpoint at $r(m)$ be c^x_p. p must take an additional checkpoint in the following two cases [6].

(Rule 2-1) There is a checkpoint c (in a process other than p) satisfying $c^x_p \rightarrow c$ and $c \rightarrow r(m)$.

(Rule 2-2) After c^x_p, p sends a message to a process which does not know the new GCN (Fig. 3).

The formal proof of the correctness of the rules is shown in [6]. Intuitively, the necessity of an additional checkpoint is as follows: since there is an initiation c_0 whose GCN is y and $c_0 \rightarrow r(m)$, any checkpoint c_1 after $r(m)$ satisfies $c_0 \rightarrow c_1$ and c_1 cannot be selected as the checkpoint for the consistent global checkpoint whose GCN is y. Thus, if p does not take an additional checkpoint before $r(m)$, p must

set c_p^x or a checkpoint before c_p^x as the checkpoint for GCN y.

For (Rule 2-1), there is an initiation c' satisfying $gc(c', q) = c$ for some process q. If $gc(c', p)$ is c_p^x or a checkpoint before c_p^x, $c_p^x \rightarrow c$ and $gc(c')$ is not consistent. Thus, an additional checkpoint is necessary.

For (Rule 2-2), assume that p sends a message to q after c_p^x. If q initiates checkpoints some time after the receive event, there might be an initiation c' whose GCN is y. Thus, the global checkpoint for y is not consistent since $c_p^x \rightarrow c'$ if c_p^x or a checkpoint before c_p^x is selected. Thus, an additional checkpoint is necessary.

First consider (Rule 2-1) in distributed mobile systems. There are four cases: (1) p is an MSS and c is in an MSS; (2) p is an MSS and c is in an MH; (3) p is an MH and c is in an MSS; and (4) p is an MSS and c is in an MH. Case (1) is identical to the one for the system without MHs in [6]. The outline of checking (Rule 2-1) is as follows.

s_i's variable $ck_i(j)$ has the number of checkpoints. $ck_i(j) = x(\geq 0)$ if $c_{s_j}^x \rightarrow e$ is satisfied, where e is s_i's current event. $ck_i(j) = -1$ if $\perp_{s_j} \not\rightarrow e$. $ck_i(i)$ is s_i's newest checkpoint number. In Fig. 2, $ck_2(1) = 1$ and $ck_2(2) = 2$ at $c_{s_2}^2$ in process s_2.

Boolean variable $see_i(j)$ has information about the new checkpoint. $see_i(j) = true$ if s_i knows that there is a checkpoint c satisfying $c_{s_j}^x \rightarrow c$, where $c_{s_j}^x$ is s_j's newest (the $ck_i(j)$-th) checkpoint. If there is no such checkpoint c, $see_i(j) = false$. In Fig. 2, $see_2(1) = true$ at $c_{s_2}^2$ in process s_2.

If $see_i(i) = true$ at s_i's event e, the following condition is satisfied. For s_i's newest checkpoint $c_{s_i}^x$, there is a checkpoint c which satisfies $c_{s_i}^x \rightarrow c$ and $c \rightarrow e$. Thus in order to detect (Rule 2-1) between the MSSs, the following condition must be tested.

(Condition 2-1-1:MSS) Take an additional checkpoint if $see_i(i) = true$.

The detail of the update rule of the variables are shown in the algorithm in Fig. 4. Its outline for ck is as follows: $ck_i(i)$ is incremented if a checkpoint is taken in s_i. When s_i sends a message m, the current value of ck_i is piggybacked on m. When s_i receives a message m from s_j, let the value of ck_j on m be mck. Set $ck_i(k) := max(ck_i(k), mck(k))$ for each k.

The outline of the update rule for see is as follows: When s_i takes a checkpoint, set $see_i(j) := true$ for all $j(\neq i)$ and $see_i(i) := false$. When s_i receives a message m from s_j, let the value of see_j and ck_j piggybacked on m be $msee$ and mck. For each k, execute the following. If $mck(k) > ck_i(k)$, s_j knows s_k's newer checkpoint. Thus, $see_i(k) := msee(k)$ is executed. If $mck(k) < ck_i(k)$, s_i knows s_k's newer checkpoint. Thus, $see_i(k)$ is unchanged. If $mck(k) = ck_i(k)$, both processes know the same checkpoint in s_k. Thus, set $see_i(k) := see_i(k) \vee msee(k)$.

Next let us discuss case (3)(4).

Theorem 1 *It is unnecessary for MH h to check whether (Rule 2-1) is satisfied for checkpoint c in processes other than its current MSS.* ■

(Proof) Suppose that (Rule 2-1) is satisfied for a checkpoint in other than its current MSS. There is a causal sequence $CS = c_h^x, s(m_0), r(m_0), \ldots, c, \ldots, s(m), r(m)$, where c_h^x is MH h's current checkpoint at $r(m)$. If a handoff occurs between c_h^x and $r(m)$, an additional checkpoint is taken between these events and c_h^x is not h's current checkpoint at $r(m)$. Thus, a handoff does not occur and $r(m_0)$ and $s(m)$ are executed in the same MSS s_i. Since c is not in s_i, CS can be written as follows. $CS = c_h^x, s(m_0), r(m_0), s(m_1), r(m_1), \ldots, c, \ldots, s(m'), r(m'), s(m), r(m)$, where $s(m_1)$ and $r(m')$ are s_i's events. Suppose that there is no s_i checkpoint between $r(m_0)$ and $r(m')$. Let s_i's current checkpoint at $r(m')$ be $c_{s_i}^y$. Since $c_{s_i}^y$ is before $r(m_0)$, $c_{s_i}^y \rightarrow c$ and $c \rightarrow r(m')$ and thus s_i satisfies the condition in (Rule 2-1). Therefore, s_i takes an additional checkpoint just before $r(m')$. This contradicts the fact that there is no checkpoint between $r(m_0)$ and $r(m')$. Thus this case cannot occur.

Now consider the case when there is a checkpoint $c_{s_i}^y$ in s_i between $r(m_0)$ and $r(m')$. Since $c_h^x \rightarrow c_{s_i}^y$ and $c_{s_i}^y \rightarrow r(m)$, the condition in (Rule 2-1) is satisfied for s_i's current MSS s_i. ■

Thus MH h just needs to check the satisfaction of (Rule 2-1) between current MSS. This is achieved as follows. When MH h is connected to MSS s_i, s_i maintains a variable $hsee(h)$. $hsee_i(h) = x$ if $c_h^x \rightarrow c$ is satisfied for MH h's checkpoint c_h^x, where c is s_i's current checkpoint. When message m is sent from s_i to h, $hsee$ is piggybacked on m. Let $mhsee$ be the value of $hsee_i(h)$ piggybacked on m. MH h maintains variable ck_h which has h's current checkpoint number (just as $ck_i(i)$ in MSS s_i). In Fig. 2, $hsee_2(h) = 1$ at $c_{s_2}^2$ in process s_2. $ck_h = 2$ at $r(m_4)$ in process h.

Then the condition satisfying (Rule 2-1) for MH is written as follows:

(Condition 2-1-2:MH) Take an additional checkpoint if $ck_h = mhsee$.

Lastly, consider case (2). This rule is similar to the one in (Condition 2-1-2).

Theorem 2 *It is unnecessary for MSS s_i to check whether (Rule 2-1) is satisfied for checkpoint c other than handoff checkpoints in MH it does not support now.* ■

(Proof) Suppose that (Rule 2-1) is satisfied for a checkpoint c other than handoff checkpoints in MH h which s_i does not support now. There is a causal sequence $CS = c_{s_i}^x, s(m_0), r(m_0) \ldots, s(m_k), r(m_k), c, s(m_{k+1}),$

$r(m_{k+1}), \ldots, s(m), r(m)$, where $c^x_{s_i}$ is s_i's current checkpoint at $r(m)$. If the processes executing $s(m_k)$ and $r(m_{k+1})$ are different, a handoff occurs in h between $s(m_k)$ and $r(m_{k+1})$. Thus, there is a causal sequence $c^x_{s_i}, s(m_0)$, $r(m_0) \ldots, s(m_k), r(m_k), c^H, s(m_{k+1}), r(m_{k+1}), \ldots,$ $s(m), r(m)$, where c^H is a handoff checkpoint. Hence, (Rule 2-1) is also satisfied for a handoff checkpoint.

Now consider no handoff occurs between $r(m_k)$ and $s(m_{k+1})$. Thus $s(m_k)$ and $r(m_{k+1})$ are executed by the same process, h's current MSS. Let s_j be the MSS. Suppose that there is no s_j's checkpoint between $s(m_k)$ and $r(m_{k+1})$. Let s_j's current checkpoint at $r(m_{k+1})$ be $c^y_{s_j}$. Since $c^y_{s_j}$ is before $s(m_k)$, $c^y_{s_j} \rightarrow c$ and $c \rightarrow r(m_{k+1})$ and thus s_j satisfies the condition in (Rule 2-1). Therefore, s_j takes an additional checkpoint just before $r(m_{k+1})$. This contradicts the fact that there is no checkpoint between $s(m_k)$ and $r(m_{k+1})$. Thus this case cannot occur.

Now consider the case when there is a checkpoint $c^y_{s_j}$ in s_j between $s(m_k)$ and $r(m_{k+1})$. Since $c^x_{s_i} \rightarrow c^y_{s_j}$ and $c^y_{s_j} \rightarrow r(m)$, the condition of (Rule 2-1) is satisfied for MSS s_j's checkpoint $c^y_{s_j}$. ■

From Theorem 2, case (2) is divided into the following two subcases. (2-1) p is an MSS and c is a checkpoint in an MH p supports now; (2-2) p is an MSS and c is a handoff checkpoint in an MH.

In order to detect case (2-1), the following variables are introduced. MH h maintains a variable see_h. $see_h = x$ if $c^x_{s_i} \rightarrow c$ is satisfied for MSS s_i's checkpoint $c^x_{s_i}$, where c is h's current checkpoint. When message m is sent from h to s_i, see is piggybacked on m. Let $msee$ be the value of see_h piggybacked on m. In Fig. 2, $see_h = -1$ at any event in h.

(Condition 2-1-3:MSS) Take an additional checkpoint if $ck_i(i) = msee$.

Case (2-2) is when there is a handoff checkpoint c satisfying $c^x_{s_i} \rightarrow c$ and $c \rightarrow r(m)$ for MSS s_i. (Note that s_i might not be the station that supports h now or supported h before.)

In order to obtain information about whether MSS s_j's checkpoints satisfy $c^x_{s_j} \rightarrow c$ for a handoff checkpoint c, h's current MSS s_i maintains variable $hlck_i(h, j)$. $hlck_i(h, j) = x$ if $ck_i(j) = x$ at s_i's last send event $s(m)$ to h. Thus s_i updates $hlck_i(h, j)$ at every send event to MH h. In Fig. 2, $hlck_2(h, 1) = 1$ at $s(m_4)$ in s_2.

s_i also maintains variable $hlgcn_i(h, j)$. $hlgcn_i(h, j) = x$ if $gcn_i(j) = x$ at s_i's last send event $s(m)$ to h. In Fig. 2, $hlgcn_2(h, 1) = 1$ and $hlgcn_2(h, 2) = 2$ at $s(m_4)$ in s_2. When a handoff occurs, $hlgcn_i(h, j)$ has each process's knowledge of the GCN that was sent to MH h. Thus, $c^x_{s_j} \rightarrow c$ is satisfied for handoff checkpoint c if $hlck_i(h, j) = x$. When MH h is connected to a new MSS s_k, s_k receives information of $hlck_i(h, j)$ and $hlgcn_i(h, j)$ from h's previous MSS s_i. (Note that for an alternative implementation, MH h can carry these values and send to the new MSS at the connection.) s_k updates $see_k(j)$ and $gcn_k(j)$ by using $hlck_i(h, j)$ and $hlgcn_i(h, j)$, because s_k might know new checkpoints by the connection. Executing this update only once for each handoff is sufficient. With this modification, checking (Condition 2-1-1) is sufficient for detecting (Rule 2-1) for handoff checkpoints.

Next consider (Rule 2-2).

Theorem 3 *It is unnecessary for MH h to check whether (Rule 2-2) is satisfied.* ■

(Proof) Suppose that (Rule 2-2) is satisfied in MH h. Let c^x_h be the current checkpoint at receive event $r(m)$ and s_i be the current MSS. $s(m)$ is executed in s_i. When (Rule 2-2) is satisfied at $r(m)$, h sends a message m' to process q after current checkpoint c^x_h. q must be s_i. Otherwise, a handoff occurs after $s(m')$ and a handoff checkpoint c^H is taken after $s(m')$. Thus, h's current checkpoint is c^H (or a checkpoint after c^H) and $s(m')$ is not after h's current checkpoint. Therefore, q is s_i. In order to satisfy the condition in (Rule 2-2), s_i's gcn at $r(m)$ must be less than the gcn received by $r(m)$. This cannot occur since m is sent by s_i. Therefore, the condition in (Rule 2-2) is not satisfied in h. ■

On the other hand, (Rule 2-2) might be satisfied in MSS s_i for MH h.

Thus, the procedure for checking (Rule 2-2) in MSS is similar to the one in [6] for systems without MHs. Its outline is as follows.

MSS s_i maintains $st_i(j)$ for each s_j and $hst_i(h)$ for $h \in H_i$. $st_i(j) = true$ if s_i sends a message to MSS s_j after s_i's current checkpoint. $hst_i(h) = true$ if s_i sends a message to MH h after s_i's current checkpoint. In Fig. 2, $st_2(1) = false$ and $hst_2(h) = true$ at $s(m_4)$ in s_2.

(Condition 2-2: MSS) Take an additional checkpoint if (1) there is a MSS s_k satisfying $gcn_i(k) < mgcn$ and $st_i(k) = true$ or (2) there is a MH h satisfying $gcn_i(h) < mgcn$ and $hst_i(h) = true$.

The algorithm is shown in Fig. 4. Each MH maintains $O(1)$ integers. The result is obtained as $cgc_p(y)$ in process p. $cgc_p(y) = x$ means that c^x_p is the checkpoint for GCN y.

4. Modification of GCN assignment

In [6], a modification of (Rule 1) is shown. It suppresses unnecessary assignments of new GCN to initiations, which is similar to the one in [1]. The obvious case when new GCN is unnecessary is that a process initiates checkpoints several times without sending or receiving to the other processes. It is unnecessary to obtain different consistent global checkpoints to these initiations if a consistent global checkpoint is obtained for the first one. The cases when new GCN is unnecessary is shown in [6]. However, introducing this

modification to distributed mobile systems greatly complicates the algorithm. Thus, here a sufficient condition to suppress assigning a new GCN to an initiation is shown.

MH h maintains a variable $sgcn$, which is the maximum GCN the current MSS knows.

(2) in (Rule 1 in MH) is changed as follows.

(2) When initiation c_h^x occurs, increment gcn_h and assign it as the GCN for c_h^x if $sgcn_h = gcn_h$ or $see_h \neq sck_h$ is satisfied.

Here, the correctness is briefly shown. Let y be the value of gcn_h just before c_h^x. Let us assume that $sgcn_h \neq gcn_h$ and $see_h = sck_h$ are satisfied. In this case, consider a global checkpoint $\{cgc_p(y) | p \in H \cup S\}$. If $cgc_h(y)$ is replaced by c_h^x, the obtained global checkpoint is also consistent. The reason is as follows: Let s_i be h's current MSS. $cgc_{s_i}(y)$ is not defined before c_h^x since $sgcn_h \neq gcn_h (= y)$. Thus, $c_h^x \rightarrow cgc_{s_i}(y)$ cannot occur because s_i selects a checkpoint before the receive event of the message of GCN y notification.

Next suppose that $cgc_{s_i}(y) \rightarrow c_h^x$. Let $c = cgc_{s_i}(y)$. $c \not\rightarrow c_h^{x-1}$ because $cgc_h(y) = c_h^{x-1}$. Thus, $see_h < sch_h$ must be satisfied just before c_h^x, because sch_h is the checkpoint number of c and $c \not\rightarrow c_h^{x-1}$. It contradicts that $see_h = sck_h$. Therefore, the replaced global checkpoint is consistent.

The rule for MSS is similar to the one for MH. MSS s_i maintains $ssee_i(j)$. $ssee_i(j) = x$ if $c_{s_j}^x \rightarrow c$ is satisfied for MSS s_j's checkpoint $c_{s_j}^x$, where c is s_i's current checkpoint.

(2) in (Rule 1 in MSS) is changed as follows.

(2) When initiation $c_{s_i}^x$ occurs, increment

gcn and make it the GCN for $c_{s_i}^x$ if (a) $gcn_i(i) = gcn_i(j)$ or $ssee_i(j) \neq ck_i(j)$ is satisfied for some j or (b) $gcn_i(i) = hgcn_i(h)$ or $hsee_i(h) \neq hck_i(h)$ is satisfied for some $h \in H_i$.

The correctness proof is the same as in the case of MH.

5. Concluding remarks

This paper discussed a coordinated checkpointing algorithm for distributed mobile systems whose message overhead is independent of the number of mobile hosts. The number of checkpoints is minimized under two assumptions. Open problems includes algorithms without handoff checkpoints.

Acknowledgment The author would like to thank Dr. Hirofumi Katsuno of NTT (currently of Tokyo Denki Univ.) and Dr. Yoshifumi Ooyama of NTT for their encouragement and suggestions.

References

[1] Baldoni, R., Quaglia, F., and Fornara, P.: "An Index-Based Checkpointing Algorithm for Autonomous Distributed Systems," Proc. 16th Int. Symp. on Reliable Distributed Systems (1997).

[2] Alagar, S. and Venkatesan, S.: "Causal Ordering in Distributed Mobile Systems," IEEE Trans. on Computers, Vol. 46, No. 3, pp.353–361 (Mar. 1997).

[3] Chandy, K.M. and Lamport, L.: "Distributed Snapshots: Determining Global States of Distributed Systems," ACM Transaction on Computer Systems, Vol. 3, No. 1, pp. 63–75 (Feb. 1985).

[4] Helary, J.-M., Mostefaoui, A., Raynal, M., and Netzer, R.H.B.: "Preventing Useless Checkpoints in Distributed Computations," Proc. of 16th Symposium on Reliable Distributed Systems (Oct. 1997).

[5] Lamport, L.: "Time, Clocks, and the Ordering of Events in a Distributed System," Communications of ACM, Vol. 21, No. 7, pp. 558–565 (July 1978).

[6] Manabe, Y.: "A Distributed Consistent Global Checkpoint Algorithm with a Minimum Number of Checkpoints," Proc. of 12th Int. Conf. on Information Networking, pp.549-554 (Jan. 1998).

[7] Manabe, Y.: "A Distributed Consistent Global Checkpoint Algorithm with a Minimum Number of Checkpoints," Technical Report of IEICE, COMP97-6 (Apr. 1997).

[8] Netzer, R.H. and Xu, J.: "Necessary and Sufficient Conditions for Consistent Global Snapshots," IEEE Trans. on Parallel and Distributed Systems, Vol. 6, No. 2, pp. 165–169 (Feb. 1995).

[9] Ohori, C., Inoue, M., Masuzawa, T., and Fujiwara, H.: "A Causal Broadcast Protocol for Distributed Mobile Systems," Technical Report of IEICE, COMP97-79 (Jan. 1998).

[10] Prakash, R. and Singhal, M.: "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," IEEE Trans. on Parallel and Distributed Systems, Vol. 7, No. 10, pp. 1035–1048 (Oct. 1996).

[11] Sato, Y., Inoue, M., Masuzawa, T., and Fujiwara H.: "A Snapshot Algorithm for Distributed Mobile Systems," Proc. 16th Int. Conf. on Distributed Computing Systems, pp.734-743 (1996).

[12] Strom, R.E. and Yemini, S.: "Optimistic Recovery in Distributed Systems," ACM Trans. on Computer Systems, Vol. 3, No. 3, pp. 204–226 (Aug. 1985).

program CGC-MSS; /* program for MSS s_i. */
const n = ...; /* number of MSS */;
var $gcn(n)$, $ck(n)$, $lck(n)$: integer;
  $see(n)$, $st(n)$: boolean;
  H_i : set of process; /* MHs connected to s_i. */
  $hgcn(h)$, $hck(h)$, $hlck(h,n)$, $hlgcn(h,n)$, $hsee(h)$: integer;
  $hst(h)$: boolean;
  $cgc()$: integer /* output of the algorithm */

procedure checkpoint **begin**
  take a checkpoint;

```
    ck(i) := ck(i) + 1;
    for each k(≠ i) do see(k) :=true;
    see(i) :=false;
    for each k do st(k) :=false;
    for each h ∈ H_i do hst(h) :=false;
    for each h ∈ H_i do hsee(h) := hck(h);
end; /* end of procedure checkpoint */

procedure testcondition(ngcn) begin
    if gcn(i) < ngcn then begin
        if see(i) or
            (∃k, st(k) and gcn(k) < ngcn) or
            (∃h ∈ H_i, hst(h) and hgcn(h) < ngcn)
          then checkpoint;
        for each y(gcn(i) < y ≤ ngcn) do cgc(y) := ck(i);
    end;
    gcn(i) := max(gcn(i), ngcn);
end; /* end of procedure testcondition */

/* main */
initialization begin
    for each k(≠ i) do ck(k) := −1;
    ck(i) := 0;
    for each k do lck(k) := −1;
    for each k do gcn(k) := 0;
    for each k do see(k) :=false;
    for each k do st(k) :=false;
    for each h ∈ H_i do hgcn(h) := 0;
    for each h ∈ H_i do hck(h) := −1;
    for each h ∈ H_i do hsee(h) := −1;
    for each h ∈ H_i do hst(h) :=false;
    for each h ∈ H_i and k do hlck(h, k) := −1;
    for each h ∈ H_i and k do hlgcn(h, k) := −1;
end; /* end of initialization */

when s_i initiates a checkpoint begin
    checkpoint;
    gcn(i) := gcn(i) + 1
    cgc(gcn(i)) := ck(i);
end; /* end of checkpoint initiation */

when s_i sends m to MSS s_j begin
    send(m, gcn, ck, see) to s_j;
    st(j) :=true;
end; /* end of message sending */

when s_i sends m to MH h ∈ H_i begin
    send(m, gcn(i), ck(i), hsee(h)) to h;
    hst(h) :=true;
    for each k do hlck(h, k) := ck(k);
    for each k do hlgcn(h, k) := gcn(k);
end; /* end of message sending */

when s_i receives (m, mgcn, mck, msee) from MSS s_j begin
    for each k do
        if ck(k) = mck(k) then see(k) := see(k) ∨ msee(k)
        else if ck(k) < mck(k) then see(k) := msee(k);
    for each k do ck(k) := max(ck(k), mck(k));
    for each k do gcn(k) := max(gcn(k), mgcn(k));
    testcondition(mgcn(j));
    execute r(m);
end; /* end of receiving from MSS */

when s_i receives (m, mgcn, mck, msee) from MH h begin
    gcn(h) := max(gcn(h), mgcn(h));
    if gcn(i) < mgcn(h) then begin
        if msee = ck(i) or
            (∃k, st(k) and gcn(k) < mgcn(h)) or
            (∃h' ∈ H_i, hst(h') and hgcn(h') < mgcn(h'))
          then checkpoint;
        for each y(gcn(i) < y ≤ mgcn(h)) do cgc(y) := ck(i);
    end;
    hck(h) := max(hck(h), mck);
    gcn(i) := max(gcn(i), mgcn(h));
    execute r(m);
end; /* end of receiving from MH */

when s_i receives (disconnect, mgcn) from MH h begin
    hgcn(h) := max(hgcn(h), mgcn);
    testcondition(hgcn(h));
    H_i := H_i − {h};
    HI(h) := {hlck(h), hlgcn(h)}; /* Handoff information */
end; /* end of removing MH */

when s_i receives (connect, mgcn) from MH h begin
    H_i := H_i ∪ {h};
    receive (hlck(h), hlgcn(h)) from h's previous MSS;
    hsee(h) := −1;
    hst(h) :=false;
    for each k do
        if ck(k) ≤ hlck(h, k) then see(k) :=true;
    for each k do ck(k) := max(ck(k), hlck(h, k));
    for each k do hlck(h, k) := −1;
    for each k do gcn(k) := max(gcn(k), hlgcn(h, k));
    hgcn(h) := mgcn;
    testcondition(hgcn(h));
end; /* end of accepting */


program CGC-MH; /* program for MH h. */
var ck, gcn, sck , see : integer;
    cgc() : integer; /* output of the algorithm */

procedure mhcheckpoint begin
    take a checkpoint;
    ck := ck + 1;
    see := sck;
end; /* end of subroutine */

/* main */
initialization begin
    ck := 0;
    gcn := 0;
    sck := −1;
    see := −1;
end; /* end of initialization */

when h initiates a checkpoint begin
    mhcheckpoint;
    gcn := gcn + 1
    cgc(gcn) := ck;
end; /* end of checkpoint initiation */

when h sends m to MSS s_j begin
    send(m, gcn, ck, see) to s_j;
end; /* end of message sending */

when h receives (m, mgcn, mck, mhsee) from MSS h_j
begin
    if mhsee = ck then mhcheckpoint;
    for each y(gcn < y ≤ mgcn) do cgc(y) := ck;
    sck := max(sck, mck);
    gcn := max(gcn, mgcn);
    execute r(m);
end; /* end of message receiving */

when h connects to MSS s_j begin
    set current MSS be s_j;
    send (connect, gcn) to s_j;
end; /* end of connecting */

when h disconnects to MSS s_j begin
    set current MSS be null;
    send (disconnect, gcn) to s_j;
    mhcheckpoint; /* handoff checkpoint */
    see := −1;
    sck := −1;
end; /* end of disconnecting */
```

Figure 4. Checkpointing Algorithm.