アルゴリズム 54－3
（1996. 10. 17）

分散最始・最終全域チェックポイントアルゴリズム

真鍋 義文
NTT 基礎研究所

あらまし　通信路がFIFOとは限らない分散システムにおいて、一貫性のある全域チェックポイントを求める問題を考える。あるプロセスのチェックポイント開始に対する一貫性のある最始(最終)全域チェックポイントは各プロセスのチェックポイントの組で一貫性をもち、かつ各プロセスの要素以前(以降)のチェックポイントが必ず開始チェックポイントと一貫性がないという性質を持つものである。任意のプロセスの任意のチェックポイント開始に対して一貫性のある最始・最終全域チェックポイントを与えるという条件のもとで、各プロセスが取るチェックポイントの数の最小性を保証する分散チェックポイントアルゴリズムを示す。

A Distributed First and Last Consistent Global Checkpoint Algorithm

Yoshifumi Manabe
NTT Basic Research Laboratories

Abstract　Distributed checkpoint algorithms for non-FIFO communication channel distributed systems are discussed. A first consistent global checkpoint for checkpoint initiation is a set containing a checkpoint for each process in which any checkpoint before the element is not consistent with the initiation. A last consistent global checkpoint for checkpoint initiation is a set containing a checkpoint for each process in which any checkpoint after the element is not consistent with the initiation. This paper presents distributed algorithms that give a first and last consistent global checkpoint with a minimum number of checkpoints taken in each process.

1 Introduction

Distributed checkpointing is a fundamental way of achieving distributed system failure recovery [5]. It obtains a set of states as a consistent global checkpoint [7], in which no message is recorded as received in one process but not yet sent in another process.

Most former algorithms assume that communication channels are FIFO (First-In, First-Out). For example, in Chandy and Lamport's algorithm [2], an initiator process takes its checkpoint and sends a marker message to all processes. When a non-initiator process receives the marker for the first time, it takes a checkpoint for the process and sends a marker message to all processes. The set of taken checkpoints is consistent. When the communication is not FIFO, the following situation occurs: Initiator p_i takes checkpoint c_i, sends marker m, and then sends message m' to p_j. m' arrives before m. p_j receives m' and then takes checkpoint c_j by receiving m. (c_i, c_j) is not consistent since m' is sent after c_i and received before c_j.

In order to execute checkpointing in non-FIFO systems, the information for checkpointing must be piggybacked on program messages. Venkatesh et al. [9] and Baldoni et al. [1] have considered such an approach, though both groups assume FIFO. Venkatesh et al.'s algorithm deals each checkpoint initiation independently. Thus the number of additionally taken checkpoints can be large. Baldoni et al. have shown an algorithm in which any checkpoint belongs to a consistent global checkpoint. However, their algorithm takes further additional checkpoints for an additional checkpoint which is taken for an imaginary checkpoint initiation. If the initiation is known not to exist, all further additional checkpoints are unnecessary. Their algorithm does not consider the problem and the number of additional checkpoints is not minimized.

This paper describes two distributed checkpoint algorithms for non-FIFO communication distributed systems. The first gives a first consistent global checkpoint for any checkpoint initiation, in which any checkpoint before the element is not consistent with the initiation. This global checkpoint can be used for debugger rollback. The second algorithm gives a last consistent global checkpoint for any checkpoint initiation, in which any checkpoint after the element is not consistent with the initiation. This global checkpoint can be used for failure rollback. The number of checkpoints taken in each process is minimized.

Section 2 shows system model. Section 3 defines the first and last consistent global checkpoint. Section 4 and 5 show the first and last consistent global checkpoint algorithm and its proof. Section 6 shows additional procedures for failure rollback. Section 7 summarizes the paper.

2 System model

The distributed system is modeled by a set of finite processes $\{p_1, p_2, \ldots, p_n\}$ interconnected by point-to-point channels. 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.

Process p_i's execution is a sequence of events. System execution is a set containing each process execution. The events are internal events, send events, and receive events. p_i's send event puts a message to the channel from p_i to the destination process. p_i's receive event gets a message from one of the channels connected to p_i.

The "happened before($\rightarrow$)" relation between the events are defined as follows [4].

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

(1) e and e' are executed in the same process and e is before 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''. ■

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

Two special events, $\perp_i$ and $\top_i$, are defined for p_i. $\perp_i$ is an imaginary event before p_i's first event. For p_i's any event e_i, $\perp_i \rightarrow e_i$. $\top_i$ is p_i's current event if p_i is not terminated. If p_i is terminated, $\top_i$ is an imaginary event after p_i's last event. There is no event e which satisfies $e \rightarrow \perp_i$ or $\top_i \rightarrow e$.

For event e_i on p_i, two events on p_j, causal-past event, $cp_i^{e_i}(j)$, and causal-future event, $cf_i^{e_i}(j)$, are defined as follows.

Definition 2

- $cp_i^{e_i}(i) = cf_i^{e_i}(i) = e_i$.
- *$cp_i^{e_i}(j)$ is p_j's last event e_j which satisfies $e_j \rightarrow e_i$. If there is no event e_j which satisfies $e_j \rightarrow e_i$, $cp_i^{e_i}(j) = \perp_j$.*
- *$cf_i^{e_i}(j)$ is p_j's first event e_j which satisfies $e_i \rightarrow e_j$. If there is no event e_j which satisfies $e_i \rightarrow e_j$, $cf_i^{e_i}(j) = \top_j$.* ■

3 First and last consistent global checkpoint

This paper considers $\top_i$ and $\perp_i$ as checkpoints. $\perp_i$ is p_i's 0-th checkpoint. Let $c_k^{x_k}$ be p_k's x_k-th checkpoint. We sometimes denote it as x_k in subscripts if it is not ambiguous.

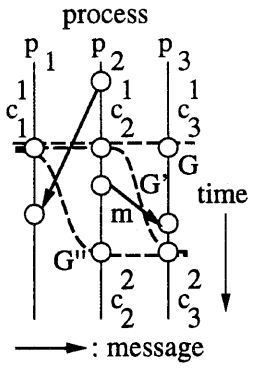

Figure 1: Consistent global checkpoint.

Definition 3 *A pair of events (e, e') is consistent if and only if $e \not\rightarrow e'$ and $e' \not\rightarrow e$.* ■

Definition 4 *A global checkpoint $(c_1, c_2, \ldots, c_n)$ is n-tuple of checkpoints where c_i is p_i's checkpoint. A global checkpoint is consistent if and only if all distinct pairs of checkpoints are consistent.* ■

Figure 1 is a system execution example. $G = \{c_1^1, c_2^1, c_3^1\}$ and $G'' = \{c_1^1, c_2^2, c_3^2\}$ are consistent, but $G' = \{c_1^1, c_2^1, c_3^2\}$ is not consistent because of message m.

If each process takes checkpoints independently, there cannot always exist a consistent global checkpoint that includes a given checkpoint initiation. Thus, each process must take some additional checkpoints. Different consistent global checkpoints can be obtained according to the additional checkpoints. The question is: What kinds of consistent global checkpoints do we need?

Two situations in which checkpoints are used are considered. The first one is recovery from process failure. Consider the case when p_i has a failure and rolls back to a checkpoint. Since the system has to roll back to a consistent state, the other processes might be forced to roll back. The additional rollback for the other processes must be as small as possible in order to minimize the overhead of re-execution. Thus, it is better for the other processes to roll back to later checkpoints.

The second one is debugging. Consider the case when an error is observed in p_i. In order to detect the bug that caused the error, p_i must roll back to the state when it is observed. p_i rolls back to a checkpoint. The other processes must roll back to a consistent state. The debugger user tries to detect the bug by examining each process. The bug might be another process p_j and a wrong message from p_j might have caused the error. If a later checkpoint is used for p_j's rollback, the bug might be hidden by p_j's further execution; for example, exiting from a subroutine and deleting all variables that decided the content of the wrong message [6]. Thus, it is

better for the other processes to roll back to former checkpoints.

For these two problems, we introduce the first and last global checkpoint.

Definition 5 *The first global checkpoint for checkpoint initiation $c_k^{x_k}$, $FG_k^{x_k}$, is defined as follows. $FG_k^{x_k}(k) = c_k^{x_k}$. For $i \neq k$, if $cp_k^{x_k}(i) = \perp_i$, $FG_k^{x_k}(i) = \perp_i$. Otherwise, $FG_k^{x_k}(i)$ is p_i's first checkpoint after $cp_k^{x_k}(i)$.* ■

Since $\top_i$ is a checkpoint, there is at least one checkpoint after $cp_k^{x_k}(i)$.

Definition 6 *The last global checkpoint for checkpoint initiation $c_k^{x_k}$, $LG_k^{x_k}$, is defined as follows. $LG_k^{x_k}(k) = c_k^{x_k}$. For $i \neq k$, if $cf_k^{x_k}(i) = \top_i$, $LG_k^{x_k}(i) = \top_i$. Otherwise, $LG_k^{x_k}(i)$ is p_i's last checkpoint before $cf_k^{x_k}(i)$.* ■

Since $\perp_i$ is a checkpoint, there is at least one checkpoint before $cf_k^{x_k}(i)$.

$LG_k^{x_k}(i)$ (or $LG_k^{x_k}(i)) = \top_i$ means that p_i need not roll back at all when p_k rolls back to $c_k^{x_k}$.

Any checkpoint of p_i before $cp_k^{x_k}(i)$ or after $cf_k^{x_k}(i)$ is not consistent with $c_k^{x_k}$. Thus, $FG_k^{x_k}$ and $LG_k^{x_k}$ are the best possible "former" and "later" global checkpoints.

$FG_k^{x_k}$ and $LG_k^{x_k}$ can be consistent by taking some additional checkpoints. Venkatesh et al.'s algorithm [9] obtains consistent $LG_k^{x_k}$ and Baldoni et al.'s algorithm [1] obtains consistent $FG_k^{x_k}$, though they did not define the concept of $LG_k^{x_k}$ or $FG_k^{x_k}$ and the number of additional checkpoints is not minimized.

In this paper, the following assumptions are made.

(1) Any process can initiate a checkpoint at any time.

(2) All information for taking checkpoints is piggybacked on program messages.

(3) p_i decides the element of $FG_k^{x_k}(i)$ (or $LG_k^{x_k}(i)$) at $cf_k^{x_k}(i)$ and it cannot be changed.

(4) The number of checkpoints other than the checkpoint initiations must be minimized.

The reason for assumption (3) is as follows. Since p_i cannot be aware of initiation $c_k^{x_k}$ before $cf_k^{x_k}(i)$, it cannot decide $FG_k^{x_k}(i)$ (or $LG_k^{x_k}(i)$) before $cf_k^{x_k}(i)$. Since rollback requests might occur at any time, the elements of $FG_k^{x_k}$ (or $LG_k^{x_k}$) must be decided as early as possible. Thus, we assume that p_i decides $FG_k^{x_k}(i)$ (or $LG_k^{x_k}(i)$) at $cf_k^{x_k}(i)$.

The present algorithms deal with checkpoint initiations and additional checkpoints differently. If a user wants to deal with an additional checkpoint as an initiation, this can be done by executing the checkpoint initiation procedure for the additional checkpoint.

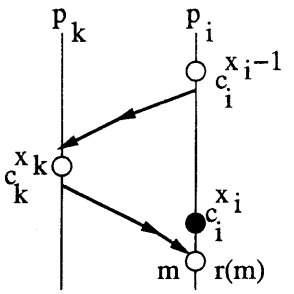

Figure 2: Rule F1.

4 First consistent global checkpoint algorithm

Throughout the paper, the value of a variable v when p_i executes event e is denoted as v_i^e.

p_i maintains a variable $ck_i(j,k)$. $ck_i(j,k) = x$ if p_i currently knows that p_j knows p_k's checkpoint c_k^x. Thus $ck_i(i,i)$ is p_i's current checkpoint number. $ck_i^e(j,k) = ck_j^{cp_i^e(j)}(j,k)$ is always satisfied. When $ck_i^e(i,j) = x$, $c_j^x \rightarrow e$ holds.

The first global checkpoint is obtained as follows. For a checkpoint initiation $c_k^{x_k}$, $FG_k^{x_k}(i) = ck_k^{x_k}(k,i) + 1(i \neq k)$. If $(ck_k^{x_k}(k,i)+1)$-th checkpoint does not exist in p_i, $FG_k^{x_k}(i) = \top_i$. In order to make it consistent, additional checkpoints must be taken.

Now consider the case when a message m from p_j arrives at p_i. Here, assume that p_i has taken $(x_i - 1)$ checkpoints before the message arrival.

The first case when x_i-th checkpoint must be taken before $r(m)$ is shown in Fig. 2. p_i knows p_k's checkpoint initiation $c_k^{x_k}$ and $ck_k^{x_k}(k,i) = x_i - 1$.

Since p_i knows the initiation, $c_k^{x_k} \rightarrow r(m)$ is satisfied. In this case, if p_i does not take x_i-th checkpoint before $r(m)$, $c_k^{x_k} \rightarrow r(m) \rightarrow c_i^{x_i} (= FG_k^{x_k}(i))$ and $FG_k^{x_k}$ is not consistent.

This condition is represented as follows. Consider a variable $ini_i(j)$. $ini_i(j) = true$ if p_i knows a checkpoint initiation c that satisfies $c_j^{x_j} \rightarrow c$ for p_j's current checkpoint $c_j^{x_j}$. The following is the rule for p_i to take a checkpoint before $r(m)$.

(Rule F1) $ini_i(i) = true$.

The updating of ini is done by sending the current value of ini in every message.

The second case is shown in Figs. 3 and 4. This case is when p_k might initiate a checkpoint after $cp_i^{r(m)}(k)$ and p_i must take x_i-th checkpoint for the initiation. Though the checkpoint is unnecessary if p_k actually does not initiate, p_i cannot predict p_k's execution after $cp_i^{r(m)}(k)$ at $r(m)$.

This case is divided into two subcases. The first case is $ck_i(k,i) = x_i - 1$. p_k initiates checkpoint $c_k^{x_k}$ just after $cp_i^{r(m)}(k)$. Assume that there is a

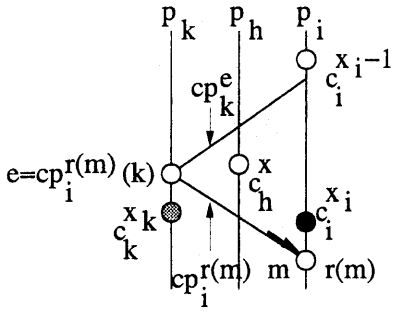

Figure 3: Rule F2 (1).

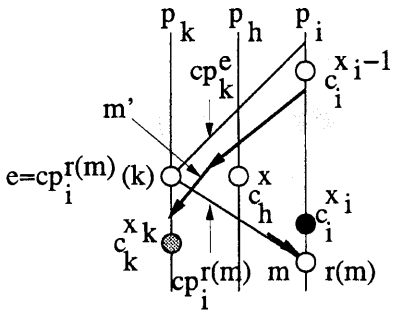

Figure 4: Rule F2 (2).

checkpoint c_h^x that satisfies $c_h^x \rightarrow r(m)$ and $c_h^x \not\rightarrow cp_i^{r(m)}(k)$. Since $c_h^x \not\rightarrow cp_i^{r(m)}(k)$, $FG_k^{x_k}(h) \leq x$. If p_i does not take x_i-th checkpoint before $r(m)$, $c_h^x \rightarrow r(m) \rightarrow c_i^{x_i} (= FG_k^{x_k}(i))$ and $FG_k^{x_k}$ is not consistent.

The second case is $ck_i(k,i) < x_i - 1$. In order for p_k to initiate a checkpoint that satisfies $FG_k^{x_k}(i) = x_i$, p_k must first receive a message that carries the information of $c_i^{x_i-1}$ and then initiate. Assume that there is a message m' that carries the information about $c_i^{x_i-1}$, and that it is sent to p_k before $cp_i^{r(m)}$ and is not received before $cp_i^{r(m)}(k)$. Assume also that p_k receives m' just after $cp_i^{r(m)}(k)$ and then initiates a checkpoint $c_k^{x_k}$. Further assume that there is a checkpoint c_h^x that satisfies $c_h^x \rightarrow r(m)$ and $c_h^x \not\rightarrow r(m')$. Since $c_h^x \not\rightarrow r(m')$, $FG_k^{x_k}(h) \leq x$. If p_i does not take x_i-th checkpoint before $r(m)$, $c_h^x \rightarrow r(m) \rightarrow c_i^{x_i}$ and $FG_k^{x_k}$ is not consistent.

This condition is represented as follows. Introduce a variable $ad_i(j,k,h)$. $ad_i(j,k,h) = x$ if $ck_k(k,h)$ becomes at least x when p_k receives a message that carries information about p_j's $ck_i(i,j)$-th checkpoint and is sent to p_k but not yet received. If p_k already knows $ck_i(i,j)$-th checkpoint, that is, $ck_i(i,j) = ck_i(k,j)$ or such a message does not exist, $ad_i(j,k,h) = -1$.

(Rule F2) $max(ck_i(k,i), ad_i(i,k,i)) = ck_i(i,i)$ and $max(ck_i(k,h), ad_i(i,k,h)) < ck_i(i,h)$ for some pair of (k,h).

Updating ad and ck is done by sending the current value of ad and ck in every message.

Algorithm $FCGC$ is shown in Fig. 7. The proof is shown below.

Theorem 1 *$FG_k^{x_k}$ is the first global consistent checkpoint for checkpoint initiation $c_k^{x_k}$.* ■

(Proof) It is obvious that $FG_k^{x_k}$ is the first global checkpoint.

Next assume that $FG_k^{x_k}$ is not consistent and there is a pair of checkpoints $(c_h^{x_h}, c_i^{x_i})$ in $FG_k^{x_k}$ which satisfies $c_h^{x_h} \rightarrow c_i^{x_i}$. Since there is no event e that satisfies $e \rightarrow \perp_i$, $x_i > 0$.

(Case 1) $i = k$.
If $c_h^{x_h} \rightarrow c_k^{x_k}$, $ck_k^{x_k}(k,h) \geq x_h$. Thus, $c_h^{x_h}$ cannot be selected for $FG_k^{x_k}(h)$.

(Case 2) $h = k$.
There is a causal sequence from $c_k^{x_k}$ to $c_i^{x_i}$. Let p_i's receive event in this causal sequence be e_i. $ck_i^{e_i}(i,i) \leq x_i - 1$ holds. From the definition of $FG_k^{x_k}$, $ck_k^{x_k}(k,i) = x_i - 1$. Since $ini_k^{x_k}(i) = true$ and $ck_k^{x_k}(k,i) = x_i - 1$, $ini_k^e(i) = true$ for every event e in the causal sequence and $ini_i^{e_i}(i) = true$. Thus from (Rule F1), p_i must have taken x_i-th checkpoint before e_i. It contradicts the definition of e_i.

(Case 3) $i \neq k$ and $h \neq k$.
There is a causal sequence from $c_h^{x_h}$ to $c_i^{x_i}$. Let p_i's receive event in this causal sequence be e_i. $ck_i^{e_i}(i,i) \leq x_i - 1$ and $c_i^{x_i-1} \rightarrow c_k^{x_k}$ holds. $ck_i^{e_i}(i,h) = x_h$ since $c_h^{x_h} \rightarrow e_i$.

First, we show that $c_i^{x_i-1}$ is before e_i. If $c_i^{x_i-1}$ is after e_i, since $c_i^{x_i-1} \rightarrow c_k^{x_k}$, $x_h^{ch} \rightarrow e_i \rightarrow c_k^{x_k}$. This cannot occur from (Case 1). Therefore, $ck_i^{e_i}(i,i) = x_i - 1$.

Next, if $c_k^{x_k}$ is before $cp_i^{e_i}(k)$, $c_k^{x_k} \rightarrow e_i \rightarrow c_i^{x_i}$ and this cannot occur from (Case 2). Thus, $c_k^{x_k}$ is after $cp_i^{e_i}(k)$.

(Case 3-1) $ck_i^{e_i}(k,i) = x_i - 1$.
Since $ck_i^{e_i}(k,i) = ck_i^{e_i}(i,i)$, $ad_i^{e_i}(i,k,h) = -1$ for all h.

Since $ck_k^{x_k}(k,h) = x_h - 1$ and $c_k^{x_k}$ is after $cp_i^{e_i}(k)$, $ck_i^{e_i}(k,h) \leq x_h - 1$. Thus, $ck_i^{e_i}(k,h) < ck_i^{e_i}(i,h)$ is satisfied and p_i must have taken x_i-th checkpoint before e_i by (Rule F2). It contradicts $ck_i^{e_i}(i,i) = x_i - 1$.

(Case 3-2) $ck_i^{e_i}(k,i) < x_i - 1$.
Since $ck_k^{x_k}(k,i) = x_i - 1$, there is a causal sequence from $c_i^{x_i-1}$ to $c_k^{x_k}$. Take one of the above causal sequences in which for each receive event in the sequence, $ck_l(l,i) < x_i - 1$ before the receive event, where p_l is the receiver process.

First, consider the case when no send event in the causal sequence is before $cp_i^{e_i}$. In this case, the first send event $s(m_1)$, which is p_i's event, is after e_i and $c_h^{x_h} \rightarrow s(m_1)$ is satisfied. Thus, $c_h^{x_h} \rightarrow c_k^{x_k}$ and this cannot occur from (Case 1). Thus, some send

events are before $cp_i^{e_i}$. Let the last one be $s(m)$ and the receiver process be $p_{k'}$.

(Case 3-2-1) $r(m)$ is before $cp_i^{e_i}(k')$.
Note that $k' \neq k$ and $k' \neq i$. From the assumption, $ck_i^{e_i}(k', i) = ck_i^{e_i}(i, i)$ and $ad_i^{e_i}(i, k', l) = -1$ for any l.

Since the next send event in the causal sequence is after $cp_i^{e_i}(k')$, $ck_k^{x_k}(k, l) \geq ck_{k'}^{cp_i^{e_i}(k')}(k, l) (= ck_i^{e_i}(k', l))$ for any l. Thus, $ck_i^{e_i}(k', h) \leq x_h - 1$ and (Rule F2) is satisfied for (k', h). Therefore, p_i must have taken x_i-th checkpoint before e_i and it contradicts $ck_i^{e_i}(i, i) = x_i - 1$.

(Case 3-2-2) $r(m)$ is after $cp_i^{e_i}(k')$.
Let the sender of m be $p_{k''}$. First, $ck_k^{x_k}(k, l) \geq ck_{k'}^{r(m)}(k', l) \geq max(ck_i^{e_i}(k', l), ck_{k''}^{s(m)}(k'', l))$ for any l. Thus, $ck_i^{e_i}(k', h) \leq x_h - 1$ and $ck_{k''}^{s(m)}(k'', h) \leq x_h - 1$ are satisfied.

In addition, since $ck_i^{e_i}(k', i) < x_i - 1$, $ck_{k''}^{s(m)}(k'', i) = x_i - 1$, and $s(m)$ is before $cp_i^{e_i}$, $ad_i^{e_i}(i, k', i) = x_i - 1$ and $ad_i^{e_i}(i, k', h) \leq x_h - 1$ because of m.

Thus, (Rule F2) is satisfied for (k', h) and p_i must have taken x_i-th checkpoint before e_i. This contradicts $ck_i^{e_i}(i, i) = x_i - 1$. ■

Theorem 2 *Every additional checkpoint taken by FCGC is necessary for obtaining the first consistent global checkpoint for a checkpoint initiation.* ■

(Proof) The proof is done by induction on the additional checkpoints. When p_i takes additional checkpoint $c_i^{x_i}$ just before a receive event e, we show that the checkpoint is necessary under the assumption that all previously taken additional checkpoints before cp_i^e are necessary.

$ck_i^e(i, i) = x_i - 1$ is satisfied before taking the checkpoint.

(Case 1) (Rule F1) is satisfied at e.
From (F1), there is a checkpoint initiation $c_k^{x_k}$ that satisfies $c_k^{x_k} \rightarrow e$ and $ck_k^{x_k}(k, i) = x_i - 1$. If p_i does not take x_i-th checkpoint before e, $c_k^{x_k} \rightarrow c_i^{x_i}$ and $FG_k^{x_k}$ is not consistent.

(Case 2) (Rule F2) is satisfied at e.

(Case 2-1) $ck_i^e(k, i) = ck_i^e(i, i)$.
p_i cannot know p_k's event sequence after cp_i^e at e. Suppose that p_k initiates checkpoint $c_k^{x_k}$ just after $cp_i^e(k)$. $FG_k^{x_k}(i) = x_i$ since $ck_i^e(k, i) = x_i - 1$. Let $FG_k^{x_k}(h)$ be x_h. $x_h = ck_i^e(k, h) + 1$.

Since $ck_i^e(k, h) < ck_i^e(i, h)$ from (F2), $c_h^{x_h} \rightarrow e$. Therefore, if p_i does not take x_i-th checkpoint before e, $c_h^{x_h} \rightarrow c_i^{x_i}$ and $FG_k^{x_k}$ is not consistent.

(Case 2-2) $ck_i^e(k, i) < ck_i^e(i, i)$.
From (F2) and the definition of ad, there is a message m sent from a process $p_{k'}$ to p_k that satisfies $ck_{k'}^{s(m)}(k', i) = ck_i^e(i, i)$ and $ck_{k'}^{s(m)}(k', h) = ad_i^e(i, k, h)$.

Suppose that p_k receives m after $cp_i^e(k)$ and then initiates a checkpoint $c_k^{x_k}$. In this case, $FG_k^{x_k}(i) = x_i$. Let $FG_k^{x_k}(h)$ be x_h. $x_h = max(ck_i^e(k, h), ad_i^e(i, k, h)) + 1 \leq ck_i^e(i, h)$. Thus, $c_h^{x_h} \rightarrow e$.

Therefore, if p_i does not take x_i-th checkpoint before e, $c_h^{x_h} \rightarrow c_i^{x_i}$ and $FG_k^{x_k}$ is not consistent. ■

The minimality of the number of additional checkpoints is shown below.

Theorem 3 *For any distributed first consistent global checkpoint algorithm A, there is a system execution in which additional checkpoints are fewer by FCGC than by A.*

(Proof) Consider a consistent global state $G = (e_1, \ldots, e_n)$ in which A and $FCGC$ execute differently about taking checkpoints at some $e_i \in G$ and equivalently at any e for $e \rightarrow e_j (j = 1, \ldots, n)$. From Theorem 2, the case when $FCGC$ takes a checkpoint and A does not take one at e_i does not occur. Thus, A takes a checkpoint and $FCGC$ does not take one at e_i. Consider the following execution after G. $p_j (j = 1, \ldots, n)$ initiates a checkpoint just before every event when p_j must take an additional checkpoint by $FCGC$. In this case, $FCGC$ takes no additional checkpoints after G and there are fewer additional checkpoints than in A. ■

5 Last consistent global checkpoint algorithm

This algorithm does not need a matrix $ck_i(j, k)$, instead, it uses a vector $ck_i(j)$ that satisfies $ck_i(j) = ck_i(i, j)$.

Consider the case when p_i realizes p_k's checkpoint initiation $c_k^{x_k}$ at receive event $cf_k^{x_k}(i)$. Let the message be m and its sender be p_j. p_i sets $LG_k^{x_k}(i)$ as p_i's latest checkpoint or takes an additional checkpoint before $r(m)$ and sets $LG_k^{x_k}(i)$ as the new one. In either case, $LG_k^{x_k}(i)$ is the last checkpoint before $cf_k^{x_k}(i)$ and $LG_k^{x_k}$ is the last global checkpoint.

To make $LG_k^{x_k}$ consistent and minimize the number of additional checkpoints, p_i must decide whether p_i takes a new checkpoint before $r(m)$. Assume that p_i has taken $(x_i - 1)$ checkpoints before the message arrival.

The first case when x_i-th checkpoint must be taken before $r(m)$ is shown in Fig. 5. This case is when $LG_k^{x_k}(i)$ has not been decided and $c_i^{x_i - 1} \rightarrow LG_k^{x_k}(h)$ for some (k, h). Since $LG_k^{x_k}(h) \rightarrow r(m)$ is satisfied, if p_i does not take a checkpoint before $r(m)$, there is no checkpoint consistent with $LG_k^{x_k}(h)$.

This condition is represented as follows.

Consider variables $lg_i(k, h)$ and $dp_i(k, h)$. $lg_i(k, h)$ has the element of $LG_k^{x_k}(h)$. $lg_i(k, k) = -1$ means p_k does not initiate a checkpoint. $lg_i(k, h) = -1$ means $LG_k^{x_k}(h)$ is unknown to p_i. $dp_i(k, h)$ has the dependency information, which means if $dp_i(k, h) = x$, $c_h^x \rightarrow c$ for some $c \in LG_k^{x_k}$.

(Rule L1) $lg_i(k, k) \neq -1$, $lg_i(k, i) = -1$, and $ck_i(i) = dp_i(k, i)$ for some k.

lg_i and dp_i are piggybacked on each message. When a message arrives, if dp(or lg) on the message is larger than the current one, the value is stored to

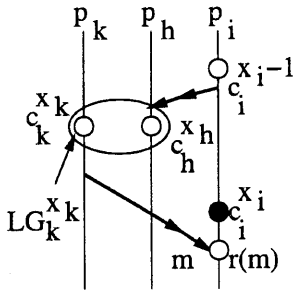

Figure 5: Rule L1.

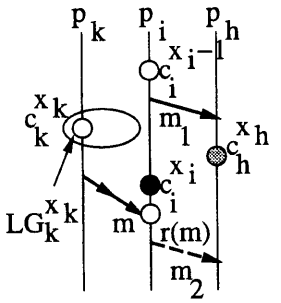

Figure 6: Rule L2.

dp_i(or lg_i). When $lg_i(k,i)$ is set to x_i, $dp_i(k,j)$ is updated by taking maximum of current $dp_i(k,j)$ and $ck_i^{x_i}(k,j)$.

The second case is shown in Fig. 6. This case is when p_i sends a message m_1 to a process p_h after $c_i^{x_i-1}$ and neither $LG_k^{x_k}(i)$ nor $LG_k^{x_k}(h)$ has been decided. This additional checkpoint is necessary if there occurs the situation in which p_i sends message m_2 to p_h after $r(m)$ and p_h executes $r(m_1)$, takes a checkpoint $c_h^{x_h}$, and then executes $r(m_2)$.

Now, since $cf_k^{x_k}(h) = r(m_2)$, $LG_k^{x_k}(h)$ is $c_h^{x_h}$ or an additional checkpoint that is taken just before $r(m_2)$. In either case, $c_i^{x_i-1} \rightarrow c_h^{x_h}$ and $LG_k^{x_k}$ is not consistent if $LG_k^{x_k}(i) = x_i - 1$. Therefore, p_i must take an additional checkpoint before $r(m)$.

This condition is represented as follows. Consider a variable $st_i(j)$. $st_i(j) = true$ if p_i has sent a message to p_j after p_i's latest checkpoint.

(Rule L2) $lg_i(k,k) \neq -1$, $lg_i(k,i) = -1$, $lg_i(k,h) = -1$, and $st_i(h) = true$ for some pair of (k,h).

$LG_k^{x_k}$ is obtained as follows. For a checkpoint initiation $c_k^{x_k}$, if $cf_k^{x_k}(i) = \top_i$, $LG_k^{x_k}(i) = \top_i$. Otherwise, $LG_k^{x_k}(i) = lg_i^{cf_k^{x_k}(i)}(k,i)$.

Note that $cf_k^{x_k}(i) = cf_k^{x_k'}(i)$ in some cases, that is, the information for two different initiations by the same process arrives to a process at the same time. In this case, $LG_k^{x_k}(i) = LG_k^{x_k'}(i)$ and these two initiations are handled by one procedure.

Algorithm *LCGC* is shown in Fig. 8. The proof is shown below.

Theorem 4 *$LG_k^{x_k}$ is the last global consistent checkpoint for checkpoint initiation $c_k^{x_k}$.*

(Proof) It is obvious that $LG_k^{x_k}$ is last global checkpoint.

Next assume that $LG_k^{x_k}$ is not consistent and there is a pair of checkpoints $(c_h^{x_h}, c_i^{x_i})$ in $LG_k^{x_k}$ that satisfies $c_h^{x_h} \rightarrow c_i^{x_i}$. Since there is no event e that satisfies $\top_h \rightarrow e$, $c_h^{x_h} \neq \top_h$ holds.

(Case 1) $h = k$.

p_i's event e_i that satisfies $c_k^{x_k} \rightarrow e_i$ must be after $cf_k^{x_k}(i)$. Since $LG_k^{x_k}(i)$ is a checkpoint before $cf_k^{x_k}(i)$, $c_k^{x_k} \rightarrow c_i^{x_i}$ is not satisfied.

(Case 2) $i = k$.

Since $c_h^{x_h} \rightarrow c_k^{x_k}$, $dp_k(k,h) \geq x_h$ at $c_k^{x_k}$ and p_k's later checkpoint initiations. Thus, $dp_h(k,h) \geq x_h$ at $cf_k^{x_k}(h)$ and p_h does not select $c_h^{x_h}$ as $LG_k^{x_k}(h)$ from (Rule L1).

(Case 3) $h \neq k$ and $i \neq k$.

Let a causal sequence from $c_h^{x_h}$ to $c_i^{x_i}$ be $c_h^{x_h}$, $s(m_1)$, $r(m_1)$, ..., $s(m_a)$, $r(m_a)$, $c_i^{x_i}$. Call a the length of the causal sequence. Without loss of generality, let the pair $(c_h^{x_h}, c_i^{x_i})$ be a pair with a shortest-length causal sequence.

Let $e_h = cf_k^{x_k}(h)$. Note that $s(m_1)$ is before e_h. If not, $c_k^{x_k} \rightarrow e_h \rightarrow s(m_1) \rightarrow c_i^{x_i}$ is satisfied and it contradicts (Case 1).

Let the receiver of m_1 be p_d. First, assume that $lg_h^{e_h}(k,d) = x_d \neq -1$, that is, $lg_h(k,d)$ has been decided at e_h. If $a \neq 1$(thus, $d \neq i$) and $c_d^{x_d}$ is before $r(m_1)$, $c_d^{x_d} \rightarrow c_i^{x_i}$ and this contradicts that $(c_h^{x_h}, c_i^{x_i})$ is a pair with a shortest-length causal sequence. Thus, $c_d^{x_d}$ is after $r(m_1)$ (or $a = 1$) and $c_h^{x_h} \rightarrow c_d^{x_d}$ holds. Since $lg_h^{e_h}(k,d) = x_d$, there is p_d's event e_d that satisfies $c_d^{x_d} \rightarrow e_d \rightarrow e_h$, $lg_d^{e_d}(k,d) = x_d$, and $lg_d^{e_d}(k,k) = lg_h^{e_h}(k,k)$. Since $c_h^{x_h} \rightarrow c_d^{x_d}$, $dp_h^{e_h}(k,h) \geq x_h$ and p_h does not select $c_h^{x_h}$ as $LG_k^{x_k}(h)$ from (Rule L1).

The remaining case is when $lg_h^{e_h}(k,d) = -1$ at e_h. In this case, since $st_h^{e_h}(d) = true$ and $lg_h^{e_h}(k,d) = -1$, p_h does not select $c_h^{x_h}$ as $LG_k^{x_k}(h)$ from (Rule L2). ■

Theorem 5 *Every additional checkpoint taken by LCGC is necessary for obtaining the last consistent global checkpoint for a checkpoint initiation.* ■

(Proof) We show that when p_i takes an additional checkpoint just before a receive event e, the checkpoint is necessary under the assumption that all previously taken additional checkpoints before cp_i^e are necessary.

Assume that p_i has taken $x_i - 1$ checkpoints. Thus $ck_i^e(i) = x_i - 1$. Let $lg_i^e(k,k) = x_k$.

(Case 1) (Rule L1) is satisfied at e.

Since $ck_i^e(i) = dp_i^e(k,i)$, $c_i^{x_i-1} \rightarrow LG_k^{x_k}(h)$ is satisfied for some h. If p_i does not take a checkpoint

before e, $LG_k^{x_k}(i) \rightarrow LG_k^{x_k}(h)$ is satisfied. Therefore, p_i must take an additional checkpoint before e.

(Case 2) (Rule L2) is satisfied at e.
Let m_1 be the message sent from p_i to p_h after $c_i^{x_i-1}$. Since $lg_i^e(k,h) = -1$, p_i does not know the element of $LG_k^{x_k}(h)$ at e. Consider the case that p_i does not take an additional checkpoint before e and $LG_k^{x_k}(i) = x_i - 1$. There is an imaginary execution after cp_i^e as follows. p_i sends message m_2 to p_h. p_h executes $r(m_1)$, takes a checkpoint c_h^x, and then executes $r(m_2)$. Note that p_h might take an additional checkpoint $c_h^{x'}$ at $r(m_2)$. In either case, $cf_k^{x_k}(h)$ is $r(m_2)$ and $LG_k^{x_k}(h)$ is c_h^x or $c_h^{x'}$. Thus, $c_i^{x_i-1} \rightarrow s(m_2) \rightarrow c_h^x (\rightarrow c_h^{x'})$ and $LG_k^{x_k}$ is not consistent. ■

The proof of the minimality of the number of additional checkpoints is exactly the same as the one for first consistent global checkpoint.

Theorem 6 *For any distributed last consistent global checkpoint algorithm A, there is a system execution in which additional checkpoints are fewer by LCGC than by A.*

6 Rollback procedures

This section shows additional procedures for rollback using LG.

The outline of a rollback state decision, which is similar to the one in [3] is as follows. When p_i initiates a rollback, p_i selects its checkpoint for rollback. Let the checkpoint be $LG_k^{x_k}(i)$, that is, it is initiated by p_k. Each process has a set R_i. Initially, $R_i = \phi$. p_i sends a message to p_k to roll back from $LG_k^{x_k}(k)$. p_k sets R_k as the set of processes to which p_k has sent a message after $LG_k^{x_k}(k)$. p_k sends rollback message ($LG_k^{x_k}$, R_k) to every process in R_k. Process p_j, which receives the rollback message ($LG_k^{x_k}$, R), calculates the set of processes, R_j, to which p_j has sent a message after $LG_k^{x_k}(j)$ and sends a rollback message ($LG_k^{x_k}$, $R \cup R_j$) to every process in $R_j - R$. With this procedure, only the processes that might need to rollback execute the rollback procedure and the other processes do nothing.

In order to re-execute after the rollback state, the messages sent and not received before the rolledback consistent global checkpoint are necessary. The message retransmission method is similar to the one in [8]. During the execution, each process saves every sending message with a sequence number for each channel. Each message contains the sequence number and an incarnation number for each channel which increments when a rollback occurs. When process p_i is rolled back to c_i, p_i informs process p_j of the sequence numbers of unreceived messages by sending the maximum sequence number $n_{i,j}$ received from p_j before c_i and missing sequence numbers before $n_{i,j}$ (since the communication is not FIFO). p_j resends messages sent before c_j and not received before c_i with a new incarnation number. p_i discards messages with an old incarnation number.

7 Conclusion

This paper showed checkpoint algorithms for obtaining a first and last consistent global checkpoint. The number of additional checkpoints is shown to be minimized. One of the remaining problems is obtaining a consistent global checkpoint, which does not have to be the first or the last, with the minimum number of additional checkpoints.

Acknowledgment The author would like to thank Dr. Hirofumi Katsuno of NTT for his encouragement and suggestions.

References

[1] Baldoni, R., Helary, J.M., Mostefaoui, A., and Raynal, M.: "Consistent Checkpointing in Message Passing Distributed Systems," INRIA Technical Report No. 2564 (June 1995).

[2] 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).

[3] Higaki, H., Shima, K., Tachikawa, T., and Takizawa, M.: "Checkpoint and Rollback in Asynchronous Distributed Systems," IPSJ Technical Report SIGDPS, 96-DPS-76, pp. 43–48 (May 1996).

[4] 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).

[5] Manabe, Y. and Aoyagi, S.: "Distributed Checkpoint and Rollback Algorithms," Journal of IPS Japan, Vol. 34, No. 11, pp. 1366–1374 (Nov. 1993) (in Japanese).

[6] Manabe, Y. and Imase, M.: "Global Conditions in Debugging Distributed Programs, " Journal of Parallel and Distributed Computing, Vol. 15, No. 1, pp. 62–69 (May 1992).

[7] 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).

[8] Sistla, A.P. and Welch, J.L.: "Efficient Distributed Recovery Using Message Logging," Proc. of 8th Symp. on Principles of Distributed Computing, pp. 223–238 (Aug. 1989).

[9] Venkatesh, K., Radhakrishnan, T., and Li, H.F.: "Optimal Checkpointing and Local Recording for Domino-Free Rollback Recovery," Information Processing Letters, Vol. 25, No. 5, pp. 295–303 (July 1987).

```
program FCGC; /* program for p_i. */
const n = ...; /* number of processes */
var ck(n,n), ad(n,n,n): integer;
  ini(n): boolean;
procedure checkpoint begin
  take a checkpoint;
  ck(i,i) := ck(i,i) + 1;
  for each k,h do ad(i,k,h) := -1;
  ini(i) :=false
end; /* end of subroutine */

/* main */
initialization begin
  for each k,h do ck(k,h) := -1;
  ck(i,i) := 0;
  for each k,h,l do ad(k,h,l) := -1;
  for each k do ini(k) :=false;
end /* end of initialization */

when p_i initiates a checkpoint begin
  checkpoint;
  for each k ≠ i do ini(k) :=true;
end /* end of checkpoint initiation */

when p_i sends m to p_j begin
  send(m, ck, ini, ad) to p_j;
  for each k do
    if ck(j,k) < ck(i,k) and ad(k,j,k) = -1
      then for each h do ad(k,j,h) := ck(i,h);
end /* end of message sending */

when message (m, mck, mini, mad)
arrives from p_j begin
  for each k do begin
    if ck(i,k) < mck(j,k) then ini(k) := mini(k);
    if ck(i,k) = mck(j,k) then
      ini(k) := mini(k) ∨ ini(k);
  end
  for each k,h do
    ck(k,h) := max(ck(k,h), mck(k,h));
  for each k do ck(i,k) := max(ck(i,k), mck(j,k));
  for each k,h do begin
    if ck(i,k) = ck(h,k) then
      for each l do ad(k,h,l) := -1
    else if ck(i,k) = mad(k,h,k) and
      ck(i,k) = ad(k,h,k) then for each l do
        ad(k,h,l) := min(ad(k,h,l), mad(k,h,l))
    else if ck(i,k) = mad(k,h,k) and
      ck(i,k) > ad(k,h,k) then
        for each l do ad(k,h,l) := mad(k,h,l)
    else if ck(i,k) > mad(k,h,k) and
      ck(i,k) > ad(k,h,k) then
        for each l do ad(k,h,l) := -1;
  end;
  if ini(i) or for some (k,h),
    (max(ck(k,i), ad(i,k,i)) = ck(i,i) and
    max(ck(k,h), ad(i,k,h)) < ck(i,h))
      then checkpoint;
  receive m;
end /* end of message arrival */
```

Figure 7: Algorithm FCGC.

```
program LCGC; /* program for p_i. */
const n = ...; /* number of processes */
var ck(n), lg(n,n), dp(n,n), ock(n): integer;
    /* ock has the value of ck
       when the latest checkpoint is taken */
  st(n): boolean;
procedure checkpoint begin
  take a checkpoint;
  ck(i) := ck(i) + 1;
  for each k do st(k) :=false;
  for each k do ock(k) := ck(k)
end; /* end of subroutine */

/* main */
initialization begin
  for each k ≠ i do ck(k) := -1;
  for each k ≠ i do ock(k) := -1;
  ck(i) := 0; ock(i) := 0;
  for each k,h do lg(k,h) := -1;
  for each k do st(k):=false;
end /* end of initialization */

when p_i initiates a checkpoint begin
  checkpoint;
  for each k ≠ i do lg(i,k) := -1;
  lg(i,i) := ck(i);
  for each k do dp(i,k) := ck(k);
end /* end of checkpoint initiation */

when p_i sends m to p_j begin
  send(m, ck, lg, dp) to p_j;
  st(j) :=true;
end /* end of message sending */

when message (m, mck, mlg, mdp)
arrives from p_j begin
  for each k do begin
    if lg(k,k) = mlg(k,k) then
      for each h do begin
        lg(k,h) := max(lg(k,h), mlg(k,h));
        dp(k,h) := max(dp(k,h), mdp(k,h));
      end
    else if lg(k,k) < mlg(k,k) then
      for each h do begin
        lg(k,h) := mlg(k,h); dp(k,h) := mdp(k,h);
      end
  end /* end of loop by k */
  if for some k, lg(k,k) ≠ -1 and lg(k,i) = -1
    and (ck(i) = dp(k,i) or
      (for some h, lg(k,h) = -1 and st(h)))
        then checkpoint;
  for each k do
    if lg(k,k) ≠ -1 and lg(k,i) = -1 then begin
      lg(k,i) := ck(i);
      for each h do
        dp(k,h) := max(dp(k,h), ock(h));
    end;
  for each k do ck(k) := max(ck(k), mck(k));
  receive m;
end /* end of message arrival */
```

Figure 8: Algorithm LCGC.