A Distributed First and Last Consistent Global Checkpoint Algorithm

Yoshifumi Manabe
NTT Basic Research Laboratories
3-1 Morinosato-Wakamiya, Atsugi-shi, Kanagawa 243-01 Japan
manabe@theory.brl.ntt.co.jp

Abstract

Distributed coordinated checkpointing algorithms are discussed. The first global checkpoint for a checkpoint initiation is a set containing the checkpoint for each process in which any checkpoint before the element is not consistent with the initiation. The last global checkpoint for a checkpoint initiation is a set containing the checkpoint for each process in which any checkpoint after the element is not consistent with the initiation. This paper presents distributed algorithms that make the first and last global checkpoints consistent with a minimum number of checkpoints taken in each process.

1 Introduction

Distributed coordinated checkpointing obtains a set of states as a consistent global checkpoint [8], in which no message is recorded as received in one process and as not yet sent in another process. It can be used for process rollback[1]. When a process initiates checkpointing, additional checkpoints must be taken in other processes in order to obtain a consistent global checkpoint that includes the initiation. Different global checkpoints might be obtained depending on the additional checkpoints taken by each process.

This paper considers two cases and defines two kinds of global checkpoint. The first case is recovery from failure. When process p_i experiences a failure, all processes roll back to each state in a consistent global checkpoint. The additional rollback for processes other than p_i 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 latter checkpoints. The second case is rollback in debugging. Assume that p_i rolls back to a state when an error is observed. The bug might be in process p_j and a wrong message from p_j might have caused the error. Thus, the debugger user wants to observe each process in a consistent global state. If a latter checkpoint is used for p_j's rollback, the bug might be hidden by further execution of p_j; for example, exiting from a subroutine and deleting all variables that decided the content of the wrong message [4]. Thus, it is better for the other processes to roll back to former checkpoints. This paper thus defines two global checkpoints: the first and last global checkpoints [5]. The first (last) global checkpoint for a checkpoint initiation is a set containing the checkpoint for each process in which any checkpoint before (after) the element is not consistent with the initiation. This paper then gives two distributed algorithms that make the first and last global checkpoint consistent with the minimum number of additional checkpoints taken in each process.

Though independent checkpointing algorithms, such as that in [9], do not need consistent global checkpoints, they can be used only for systems in which all non-deterministic events can be recorded during execution and replayed during re-execution. For systems in which records of non-deterministic events can be very large or replaying non-deterministic events is difficult, a consistent global checkpoint is necessary.

Chandy et al.'s distributed snapshot algorithm [2] obtains a consistent global checkpoint (neither the first nor the last) for concurrent initiations. The author [7] extended their algorithm and the extended version minimizes the number of additional checkpoints. Venkatesh et al.'s algorithm [10] obtains a last global checkpoint that is consistent, and Baldoni et al.'s algorithm [1] obtains a first global checkpoint that is consistent, but they did not define the concept of first and last global checkpoints. Their algorithms do not minimize the number of additional checkpoints.

2 The first and last global checkpoint

The distributed system is modeled by a finite set of processes $\{p_1, p_2, \ldots, p_n\}$ interconnected by point-to-point channels. Channels are assumed to be error-free, non-FIFO, and have infinite capacity. The communication is asyn-

[1] In order to roll back, the messages which have been sent but not received must be restored. The message restoration method is similar to that in [9] and the details are given in [6]. This paper thus discusses obtaining a consistent global checkpoint.

chronous; that is, the delay experienced by a message is unbounded but finite. p_i's execution is a sequence of p_i's events which include checkpoint initiations. Checkpoint initiations are done independently by each process. System execution E is the set of each process's executions. p_i's execution with checkpointing algorithm A is p_i's execution interleaved with the additional checkpoints taken by A in p_i. System execution with A, $E(A)$, is the set of each process's execution with A.

The following assumptions are common for distributed checkpointing algorithm A [1][10]. A has no prior knowledge about execution E. All information for A is piggybacked on program messages between processes. When p_i receives a message m, A can get the information piggybacked on m and take an additional checkpoint before p_i executes the receive event.

The "happened before($\rightarrow$)" relation between the events in $E(A)$ is defined as follows [3].

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''. ■

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

Two special events, $\perp_i$ and $\top_i$, are defined for p_i. $\perp_i$ is an imaginary event which is p_i's initial state. $\top_i$ is p_i's current event if p_i is not terminated. If p_i is terminated, $\top_i$ is an imaginary event which is p_i's terminal state. For any p_i event e_i, $\perp_i \rightarrow e_i$ and $e_i \rightarrow \top_i$ hold. This paper considers $\top_i$ and $\perp_i$ as checkpoints in E.

For p_i's event e_i in $E(A)$, 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 last event e_j in p_j that satisfies $e_j \rightarrow e_i$. If there is no event e_j satisfying $e_j \rightarrow e_i$, $cp_i^{e_i}(j) = \perp_j$.*
- *$cf_i^{e_i}(j)$ is first event e_j in p_j that satisfies $e_i \rightarrow e_j$. If there is no event e_j satisfying $e_i \rightarrow e_j$, $cf_i^{e_i}(j) = \top_j$.* ■

Intuitively, $cp_i^{e_i}(j)$ is p_j's last event which is known to p_i at e_i. $cf_i^{e_i}(j)$ is p_j's first event which knows e_i. In Fig. 1, $cp_2^{c_2^1}(1) = s(m_1)$, $cp_2^{c_2^1}(2) = c_2^1$, $cp_2^{c_2^1}(3) = s(m_2)$, $cp_2^{c_2^1}(4) = \perp_4$, $cf_2^{c_2^1}(1) = \top_1$, $cf_2^{c_2^1}(2) = c_2^1$, $cf_2^{c_2^1}(3) = r(m_5)$, and $cf_2^{c_2^1}(4) = r(m_6)$.

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

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.* ■

Definition 5 *The first global checkpoint for p_k's checkpoint initiation c_k in $E(A)$, $FG_k^{c_k}(E(A))$, is defined as follows. i-th element, $FG_k^{c_k}(E(A), i)$, is the first checkpoint in p_i which is not before $cp_k^{c_k}(i)$.* ■

$E(A)$ is omitted if it is obvious.

Definition 6 *The last global checkpoint for p_k's checkpoint initiation c_k in $E(A)$ $LG_k^{c_k}(E(A))$, is defined as follows. i-th element, $LG_k^{c_k}(E(A), i)$, is the last checkpoint in p_i which is not after $cf_k^{c_k}(i)$.* ■

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

Any checkpoint of p_i before $cp_k^{c_k}(i)$ or after $cf_k^{c_k}(i)$ is not consistent with $c_k^{c_k}$. Thus, $FG_k^{c_k}(E(A))$ and $LG_k^{c_k}(E(A))$ are the best possible "former" and "latter" global checkpoints for c_k. In Fig. 1, $FG_2^{c_2^1}(E) = (c_1^1, c_2^1, c_3^1, \perp_4)$ and $LG_2^{c_2^1}(E) = (\top_1, c_2^1, c_3^1, c_4^2)$. $FG_2^{c_2^1}(E)$ is not consistent since $c_1^1 \rightarrow c_3^1$. $LG_2^{c_2^1}(E)$ is not consistent since $c_3^1 \rightarrow c_4^2$.

Though $FG_k^{c_k}(E)$ and $LG_k^{c_k}(E)$ might not be consistent, $FG_k^{c_k}(E(A))$ and $LG_k^{c_k}(E(A))$ can be consistent by the additional checkpoints taken by A. In Fig. 1, if A takes an additional checkpoint at e_1, $FG_2^{c_2^1}(E(A), 3) = e_1$ and $FG_2^{c_2^1}(E(A))$ is consistent. If A' takes an additional checkpoint at e_2, $LG_2^{c_2^1}(E(A'), 3) = e_2$ and $LG_2^{c_2^1}(E(A'))$ is consistent. If algorithm A_0 takes an additional checkpoint just before every receive event, $FG_k^{c_k}(E(A_0))$ and $LG_k^{c_k}(E(A_0))$ is consistent for any checkpoint initiation c_k. However, the overhead of A_0 is very large. This paper shows two distributed checkpointing algorithms, FA and LA. Among algorithm A which makes every $FG_k^{c_k}(E(A))$ ($LG_k^{c_k}(E(A))$) consistent, FA (LA) takes the minimum number of additional checkpoints[2].

[2] Our 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.

3 Algorithm FA for $FG_k^{c_k}$

In the rest of the paper, a sequence number is assigned for (both of initiation and additional) checkpoints in each process in $E(A)$. $\perp_i$ is p_i's 0-th checkpoint. Let $c_k^{x_k}$ be p_k's x_k-th checkpoint. It is sometimes denoted as x_k in subscripts if it is not ambiguous.

p_i maintains a variable $ck_i(j)$. $ck_i(j) = x$ if p_i currently knows p_j's checkpoint c_j^x. $ck_i(j) = -1$ if p_i currently knows no checkpoint in p_j. If $ck_i(j) = x (\geq 0)$ at event e, $c_j^x \rightarrow e$ and $ck_j(j) = x$ at $cp_i^e(j)$ is satisfied. $ck_i(i)$ is p_i's newest checkpoint number. Updating ck can be done by sending its current value on every message. This is shown in detail in Fig. 4.

For any algorithm A, $FG_k^{c_k}(E(A))$ for p_k's checkpoint initiation c_k can be represented using ck as follows: Let $CK(i)$ be the value of $ck_k(i)$ at c_k. $FG_k^{c_k}(i) = c_i^{CK(i)+1} (i \neq k)$. If the $(CK(i)+1)$-th checkpoint does not exist in p_i, $FG_k^{c_k}(i) = \top_i$. In order to make it consistent, FA takes additional checkpoints.

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

The first case when the 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 $c_i^{x_i-1} \rightarrow c_k^{x_k}$. Since p_i knows the initiation, $c_k^{x_k} \rightarrow r(m)$ is satisfied. If p_i does not take the 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 p_i's rule for taking a checkpoint before $r(m)$.

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

The update rule of ini is shown in Fig. 4.

The second case is shown in Fig. 3. In this case p_k might initiate a checkpoint after $e_k = cp_i^{r(m)}(k)$. Though this checkpoint is unnecessary if p_k actually does not initiate after e_k, p_i takes it since p_i cannot predict p_k's execution after e_k at $r(m)$.

This case is divided into two subcases. The first subcase is when p_k knows p_i's current (the $(x_i - 1)$-th) checkpoint at e_k. p_k initiates checkpoint $c_k^{x_k}$ just after e_k. Assume that there is a checkpoint c_h^x that satisfies $c_h^x \rightarrow r(m)$ and $c_h^x \not\rightarrow e_k$. Since $c_h^x \not\rightarrow e_k$, $FG_k^{x_k}(h) \rightarrow c_h^x$ from the decision rule of FG. If p_i does not take the x_i-th checkpoint before $r(m)$, $FG_k^{x_k}(h) \rightarrow c_h^x \rightarrow r(m) \rightarrow c_i^{x_i} (= FG_k^{x_k}(i))$ and $FG_k^{x_k}$ is not consistent.

The second subcase is when p_k does not know p_i's $(x_i - 1)$-th checkpoint at e_k. 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' sent to p_k but not received before e_k, which carries the information about $c_i^{x_i-1}$. Assume also that p_k receives m' just after e_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 $c_h^x \not\rightarrow r(m')$, $FG_k^{x_k}(h) \rightarrow c_h^x$. If p_i does not take the 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 boolean variable $cr_i(j,k)$ and $ad_i(j,k,h)$. $cr_i(j,k) = true$ if p_i knows that p_j knows p_k's current (the $ck_i(k)$-th) checkpoint. Note that $cr_i(i,k)$ is always true for every k. $ad_i(j,k,h) = true$ if p_i knows that p_j will know p_h's current (the $ck_i(h)$-th) checkpoint if p_j receives any message sent to p_j that carries p_k's current (the $ck_i(k)$-th) checkpoint. If p_j already knows p_k's current checkpoint or such a message does not exist, $ad_i(j,k,h) = false$.

(Rule F2) ($cr_i(k,i) = true$ and $cr_i(k,h) = false$) or ($ad_i(k,i,i) = true$ and $cr_i(k,h) = false$ and $ad_i(k,i,h) = false$) for some pair of (k,h).

The algorithm FA, which includes the update rule of the above variables, is shown in Fig. 4.

Theorem 1 *Every additional checkpoint taken by FA is necessary.* ■

Theorem 1 is obvious from the above discussion.

Theorem 2 *$FG_k^{x_k}(E(FA))$ is consistent for any checkpoint initiation $c_k^{x_k}$.* ■

(Proof) Assume that $FG_k^{x_k}$ is not consistent and $c_h^{x_h} (= FG_k^{x_k}(h)) \rightarrow c_i^{x_i} (= FG_k^{x_k}(i))$. From the FG decision rule, $c_j^{x_j-1} \rightarrow c_k^{x_k}$ and $c_j^{x_j} \not\rightarrow c_k^{x_k}$ if $j \neq k$.

(**Case 1**: $i = k$) $c_h^{x_h} \rightarrow c_k^{x_k}$ contradicts the above fact.

(**Case 2**: $h = k$) Let the last message on the causal sequence from $c_k^{x_k}$ to $c_i^{x_i}$ be m. Since $c_i^{x_i-1} \rightarrow c_k^{x_k}$ and $c_k^{x_k} \rightarrow r(m)$, $ini_i(i) = true$ at $r(m)$. From Rule F1, p_i must have taken the x_i-th checkpoint before $r(m)$. This contradicts the assertion that $c_i^{x_i}$ is after $r(m)$.

(**Case 3**: $i \neq k$ and $h \neq k$) There is a causal sequence CS from $c_i^{x_i-1}$ to $c_k^{x_k}$. Let the sequence of messages in CS be $M_1, M_2, \ldots, M_l$. Let the process that executes $r(M_a)$ be p_{z_a} $(a = 1, \ldots, l)$. Without loss of generality, p_{z_a} satisfies $ck_{z_a}(i) < x_i - 1$ before $r(M_a)$ $(a = 1, \ldots, l)$. Note that $c_h^{x_h} \not\rightarrow e$ for any event e in CS. Otherwise, $c_h^{x_h} \rightarrow c_k^{x_k}$ and this contradicts the FG decision rule. Let the last event e in CS that satisfies $e \rightarrow c_i^{x_i}$ be e'_j on p_j. Such an event always exists because $s(M_1) \rightarrow c_i^{x_i}$. If $s(M_1)$ is after $c_i^{x_i}$, $c_i^{x_i} \rightarrow c_k^{x_k}$ and this contradicts the FG decision rule. Let $e_j = cp_i^{x_i}(j)$, that is, let e_j be p_j's last event known to p_i at $c_i^{x_i}$. From the definition, $e'_j \rightarrow e_j$.

(**Case 3-1**: e'_j is receive event $r(M)$) From the assumption, the next send event in CS is after e_j. There is a causal sequence from e_j to $c_i^{x_i}$. Let $r(m')$ be the last receive event in the causal sequence. Since $ck_j(i) = x_i - 1$ and $ck_j(h) < x_h$ at e_j, $cr_i(j,i) = true$ and $cr_i(j,h) = false$ at $r(m')$. Thus, p_i must have taken the x_i-th checkpoint before $r(m')$ from Rule F2.

(**Case 3-2**: e'_j is send event $s(M)$) Let the receiver of M be p_g and let $e_g = cp_i^{x_i}(g)$. From the assumption, $r(M)$ is after e_g. $ad_i(g,i,i) = true$, $cr_i(g,h) = false$, and $ad_i(g,i,h) = false$ are satisfied at $r(m')$ since $ck_g(i) < x_i - 1, ck_g(h) < x_h$ at e_g and $ck_g(i) = x_i - 1, ck_g(h) < x_h$ at $r(M)$. Thus, p_i must have taken the x_i-th checkpoint before $r(m')$ from Rule F2. ■

The information piggybacked on each message and kept in each process is $O(n)$ integer and $O(n^3)$ boolean values.

4 Algorithm LA for $LG_k^{c_k}$

Consider the case when p_i realizes p_k's initiation $c_k^{x_k}$ at receive event $cf_k^{x_k}(i)$. Let the message be m and its sender be p_j. Checkpointing algorithm A sets $LG_k^{x_k}(E(A), i)$ as p_i's newest checkpoint or takes a new additional checkpoint before $r(m)$ and sets $LG_k^{x_k}(E(A), i)$ as the new one. In either case, $LG_k^{x_k}(E(A), i)$ is the last checkpoint that is not after $cf_k^{x_k}(i)$. LA must decide whether it takes a new additional checkpoint before $r(m)$ in order to make $LG_k^{x_k}$ consistent with the minimum number of additional checkpoints. Assume that p_i has taken $(x_i - 1)$ checkpoints before the arrival of m.

The cases when the x_i-th checkpoint must be taken before $r(m)$ are shown in Fig. 5. The first case is when there is a checkpoint $c_h^{x_h}$ such that $c_i^{x_i-1} \to c_h^{x_h}$ and $c_h^{x_h} \to r(m)$ are satisfied. There is an initiation $c_k^{x_k}$ that satisfies $LG_k^{x_k}(h) = c_h^{x_h}$. $LG_k^{x_k}$ is not consistent if $c_i^{x_i}$ is not taken before $r(m)$ because $c_i^{x_i-1} \to c_h^{x_h}$. Thus, an additional checkpoint is necessary.

This condition is represented as follows. Consider a variable $see_i(j)$. $see_i(j) = true$ if p_i knows a checkpoint c that satisfies $c_j^{x_j} \to c$ for p_j's current checkpoint $c_j^{x_j}$.

(Rule L1) $see_i(i) = true$.

The second case is when p_i sends a message m' to 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 p_i sends message m'' to p_h after $r(m)$ and p_h executes $r(m')$, takes a checkpoint $c_h^{x_h}$, and then executes $r(m'')$. Now, since $cf_k^{x_k}(h) = r(m'')$, $LG_k^{x_k}(h)$ is $c_h^{x_h}$ or an additional checkpoint that is taken just before $r(m'')$. In either case, $c_i^{x_i-1} \to LG_k^{x_k}(h)$ 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 case is represented as follows. Variable $num_i(j) = x$ if p_i knows the newest initiation by p_j is c_j^x. Variable $dc_i(j,k) = true$ if p_i knows that p_k has decided the element of $LG_j^x(k)$, where $x = num_i(j)$. Variable $st_i(j) = true$ if p_i has sent a message to p_j after p_i's latest (the $ck_i(i)$-th) checkpoint.

(Rule L2) $dc_i(k,i) = false$, $dc_i(k,h) = false$, and $st_i(h) = true$ for some pair of (k,h).

Algorithm LA, which includes the rule for updating these variables, is shown in Fig. 6. $LG_k^{x_k}(i)$ is stored in variable lg_i. $lg_i(k, x_k) = y$ if $LG_k^{x_k}(i) = c_i^y$. Note that $lg_i(k, x_k)$ might be undefined for two reasons. First, p_i does not know initiation c_k^x, that is, $cf_k^{x_k}(i) = \top_i$ at a given state. In such a case, $LG_k^{x_k}(i) = \top_i$. The second case is when $cf_k^{x_k}(i) = cf_k^{x'_k}(i)$, that is, the information for two different initiations by p_k arrives at p_i at the same time. The information about old initiations is discarded because $LG_k^{x_k}(i) = LG_k^{x'_k}(i)$. Thus, if $lg_i(k,x)$ is undefined for every $x (x_0 < x < x_k)$ and $lg_i(k, x_k)$ is defined, $lg_i(k,x) = lg_i(k,x_k)$ for any initiation $c_k^x (x_0 < x < x_k)$.

Theorem 3 *Every additional checkpoint taken by LA is necessary.* ■

Theorem 3 is obvious from the above discussion.

Lemma 1 *[7] If a global checkpoint $(c_1, c_2, \dots, c_n)$ is not consistent, there is a pair (c_i, c_j) such that there is a causal sequence from c_i to c_j whose length is 1.* ■

Theorem 4 *$LG_k^{x_k}(E(LA))$ is consistent for any checkpoint initiation $c_k^{x_k}$.* ■

(Proof) Assume that $LG_k^{x_k}$ is not consistent. From Lemma 1, assume that $c_h^{x_h} (= LG_k^{x_k}(h)) \to c_i^{x_i} (= LG_k^{x_k}(i))$ and let the message in the causal sequence be m.

Let $e_j^{x_k}$ be the event when p_j decides $LG_k^{x_k}(j)$. Note that $LG_k^{x_k}(j)$ might be decided by deciding $LG_k^{x'_k}(j)$ for $x'_k > x_k$ as described above. In such a case, $e_j^{x_k} = e_j^{x'_k}$. $e_j^{x_k}$ is an initiation, a receive event, or $\top_j$ (in this case, $LG_k^{x_k}(j) = \top_j$). If $e_j^{x_k}$ is a receive event, $LG_k^{x_k}(j)$ is before $e_h^{x_k}$. In the other cases, $e_j^{x_k} = LG_k^{x_k}(j)$. Thus, $LG_k^{x_k}(j) \to e_j^{x_k}$ is satisfied. Note that $LG_k^{x_k}(j)$ is p_j's newest checkpoint at $e_j^{x_k}$. $num_j(k) < x_k$ is satisfied before $e_j^{x_k}$ and $num_j(k) \geq x_k$ is satisfied after $e_j^{x_k}$.

$c_h^{x_h} \neq \top_h$ since there is an event $s(m)$ after $c_h^{x_h}$.

(**Case 1**: $e_i^{x_k} \to e_h^{x_k}$) Since $c_h^{x_h}$ is p_h's newest checkpoint at $e_h^{x_k}$ and $c_h^{x_k} \to c_i^{x_i} \to e_i^{x_k} \to e_h^{x_k}$, $see_i(i) = true$ at $e_h^{x_k}$. Thus, $c_h^{x_h}$ must be the newly taken checkpoint just before $e_h^{x_k}$ from Rule L1. This contradicts the notion that there is an event $s(m)$ between $c_h^{x_h}$ and $e_h^{x_k}$.

(**Case 2**: $e_h^{x_k}$ is before $s(m)$) $num_i(k) \geq x_k$ must be satisfied at $r(m)$. Thus, $e_i^{x_k}$ must be equal to or before $r(m)$. This contradicts the notion that $c_i^{x_i}$ is after $r(m)$.

(**Case 3**: $e_i^{x_k} \not\to e_h^{x_k}$ and $e_h^{x_k}$ is after $s(m)$) Since $e_i^{x_k} \not\to e_h^{x_k}$, $dc_h(k,i) = false$ at $e_h^{x_k}$. Since there is an event $s(m)$ between $c_h^{x_h}$ and $e_h^{x_k}$, $e_h^{x_k}$ is a receive event. Let $X = num_h(k)$ at $e_h^{x_k}$. $X \geq x_k$ is satisfied. Since $st_h(i) = true$

and $dc_h(k,i) = false$ at $e_h^{x_k}$, $c_h^{x_h}$ must be the newly taken checkpoint just before $e_h^{x_k}$ from Rule L2. This contradicts the notion that there is an event $s(m)$ between $c_h^{x_h}$ and $e_h^{x_k}$. ■

The information piggybacked on each message and kept in each process (other than the output) is $O(n)$ integer and $O(n^2)$ boolean values.

Here, the rule for removing old checkpoints is shown. The amount of stable storage usage becomes large if old checkpoints are not removed. Assume that p_k rolls back to the newest initiation by p_k. p_i might be forced to roll back to $lg_i(k, num(k))$. Thus, p_i does not need the checkpoints before $min_k lg_i(k, num(k))$ and these checkpoints can be removed.

5 Conclusion

This paper showed two distributed algorithms that make the first and last global checkpoint consistent with a minimum number of additional checkpoints taken in each process. Remaining problems include reducing the amount of information used by the algorithms.

Acknowledgments 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] 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).

[4] 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).

[5] Manabe, Y.: "A Distributed First and Last Consistent Global Checkpoint Algorithm," IPSJ SIG Notes, AL96-54-3 (Oct. 1996).

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

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

[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] 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).

[10] 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).

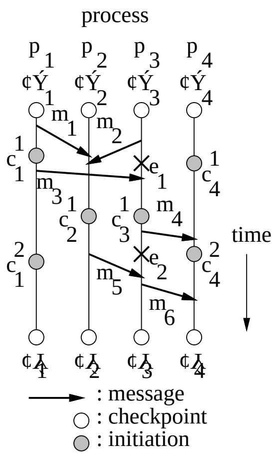

Figure 1. System execution E.

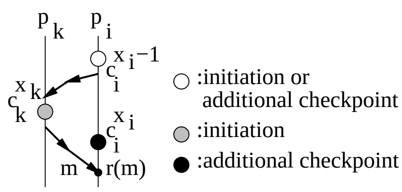

Figure 2. Rule F1.

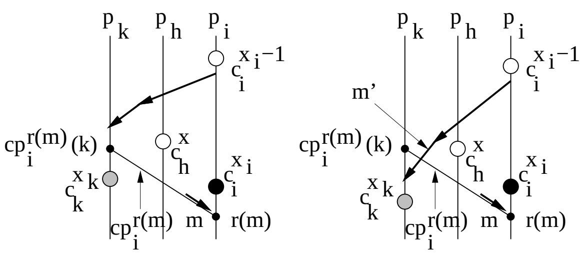

Figure 3. Rule F2.

program FA; /* program for p_i. */
const $n = \ldots$; /* number of processes */
var $ck(n)$: integer;
$ini(n)$, $cr(n,n)$, $ad(n,n,n)$: boolean;
procedure checkpoint **begin**
take a checkpoint;
$ck(i) := ck(i) + 1$;
$ini(i)$:=false;
for each $k(\neq i)$ **do** $cr(k,i)$:=false;
for each $k(\neq i), h$ **do** $ad(k,i,h)$:=false;
for each $k(\neq i), h$ **do** $ad(k,h,i)$:=false;
end; /* end of subroutine */
/* main */
initialization begin
for each $k(\neq i)$ **do** $ck(k) := -1$;
$ck(i) := 0$;
for each k **do** $ini(k)$:=false;
for each $k(\neq i), h$ **do** $cr(k,h)$:=false;
for each k **do** $cr(i,k) := true$;
for each k, h, l **do** $ad(k,h,l)$:=false;
end /* end of initialization */
when p_i initiates a checkpoint **begin**
checkpoint;
for each $k \neq i$ **do** $ini(k)$:=true;
end /* end of checkpoint initiation */
when p_i sends m to p_j **begin**
send(m, ck, ini, cr, ad) to p_j;
for each k **do**
if not($cr(j,k)$) **and not**($ad(j,k,k)$) **then**
for each h **do** $ad(j,k,h)$:=true;
end /* end of message sending */
when message (m, mck, $mini$, mcr, mad) arrives from p_j
begin
for each k **do begin**
if $ck(k) < mck(k)$ **then begin**
$ini(k) := mini(k)$;
for each $h(\neq i)$ **do** $cr(h,k) := mcr(h,k)$;
for each $h(\neq i), l$ **do**
if $mck(l) \geq ck(l)$ **then** $ad(h,k,l) := mad(h,k,l)$
else $ad(h,k,l)$:=false;
end /* end of case $ck(k) < mck(k)$ */
else if $ck(k) = mck(k)$ **then begin**
$ini(k) := ini(k) \vee mini(k)$;
for each $h(\neq i)$ **do** $cr(h,k) := cr(h,k) \vee mcr(h,k)$;
for each $h(\neq i), l$ **do**
if $mck(l) > ck(l)$ **then**
if $ad(h,k,k)$ **then** $ad(h,k,l)$:=false
else $ad(h,k,l) := mad(h,k,l)$
else if $mck(l) = ck(l)$ **then**
if ($cr(h,l)$ **or** mcr(h,l) **or**
($ad(h,k,k)$ **and not**($ad(h,k,l)$)) **or**
($mad(h,k,k)$ **and not**($mad(h,k,l)$)))
then $ad(h,k,l)$:=false
else $ad(h,k,l) := ad(h,k,l) \vee mad(h,k,l)$
else /* $mck(l) < ck(l)$ */
if $mad(h,k,k)$ **then** $ad(h,k,l)$:=false;
end /* end of case $ck(k) = mck(k)$ */
else /* $ck(k) > mck(k)$ */
for each $h(\neq i), l$ **do**
if $mck(l) > ck(l)$ **then** $ad(h,k,l)$:=false
end; /* end of if statement */
end; /* end of loop by k. */
for each $k(\neq i)$ **do** $ck(k) := max(ck(k), mck(k))$;
if $ini(i)$ **or**
$\exists(k,h)$, (($cr(k,i)$ **and not**($cr(k,h)$)) **or**
($ad(k,i,i)$ **and not**($cr(k,h)$) **and not**($ad(k,i,h)$)))
then checkpoint;
execute $r(m)$;
end /* end of message arrival */

Figure 4. Algorithm FA.

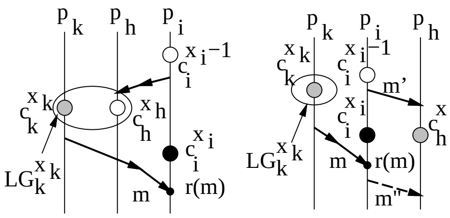

Figure 5. Rule L1 and L2.

program LA; /* program for p_i. */
const $n = \ldots$; /* number of processes */
var $ck(n)$, $num(n)$, $lg(n,*)$: integer;
$dc(n,n)$, $see(n)$, $st(n)$: boolean;
procedure checkpoint **begin**
take a checkpoint;
$ck(i) := ck(i) + 1$;
for each k **do** $st(k)$:=false;
for each $k(\neq i)$ **do** $see(k)$:=true;
$see(i)$:=false
end; /* end of subroutine */
/* main */
initialization begin
for each $k \neq i$ **do** $ck(k) := -1$;
$ck(i) := 0$;
for each k **do** $num(k) := 0$;
for each k, h **do** $dc(k,h)$:=true;
for each k **do** $see(k)$:=false;
for each k **do** $st(k)$:=false;
end /* end of initialization */
when p_i initiates a checkpoint **begin**
checkpoint;
$num(i) := ck(i)$;
for each $k(\neq i)$ **do** $dc(i,k)$:=false;
end /* end of checkpoint initiation */
when p_i sends m to p_j **begin**
send(m, ck, num, see, dc) to p_j;
$st(j)$:=true;
end /* end of message sending */
when message (m, mck, $mnum$, $msee$, mdc)
arrives from p_j **begin**
for each k **do begin**
if $num(k) < mnum(k)$ **then**
for each h **do** $dc(k,h) := mdc(k,h)$;
else if $num(k) = mnum(k)$ **then**
for each h **do** $dc(k,h) := dc(k,h) \vee mdc(k,h)$;
end /* end of loop by k */
for each k **do** $num(k) := max(num(k), mnum(k))$;
for each k **do**
if $ck(k) < mck(k)$ **then** $see(k) := msee(k)$
else if $ck(k) = mck(k)$
then $see(k) := see(k) \vee msee(k)$;
for each k **do** $ck(k) := max(ck(k), mck(k))$;
if $see(i)$ **or**
$\exists(k,h)$, (**not**($dc(k,i)$) **and not**($dc(k,h)$) **and** $st(h)$)
then checkpoint;
for each k **do**
if not($dc(k,i)$) **then begin**
$dc(k,i)$:=true;
$lg(k, num(k)) := ck(i)$;
end;
execute $r(m)$;
end /* end of message arrival */

Figure 6. Algorithm LA.