A Distributed Consistent Global Checkpoint Algorithm with a Minimum Number of Checkpoints

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

Abstract

A distributed coordinated checkpointing algorithm is shown. 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. This algorithm obtains a consistent global checkpoint for any checkpoint initiation by any process. Under Chandy and Lamport's assumption that one consistent global checkpoint is obtained for a set of concurrent checkpoint initiations, the total number of checkpoints is minimized. This paper then modifies the assumption in order to reduce the number of checkpoints further.

1 Introduction

Distributed coordinated checkpointing obtains a set of states as a consistent global checkpoint [7], 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. Since a checkpoint saves all the information necessary for rollback in stable storage, the checkpointing overhead is usually very large and the number of additional checkpoints must be minimized. This paper discusses minimizing the number of additional checkpoints by reusing the checkpoints in a consistent global checkpoint for initiation c as the checkpoints for another initiation c' [5].

Chandy and Lamport's distributed snapshot algorithm [1] obtains one consistent global checkpoint for a set of concurrent checkpoint initiations. However, every non-initiator process is forced to take a checkpoint. This paper shows two distributed checkpointing algorithms. The first one takes the minimum number of additional checkpoints under Chandy and Lamport's assumption that one consistent global checkpoint is obtained for a set of concurrent initiations. Thus among the extensions of the distributed snapshot algorithm, this algorithm is the optimal one[2]. The second algorithm in this paper modifies Chandy and Lamport's assumption in order to further reduce the number of additional checkpoints.

Prakash and Singhal [8] use a bit vector in order to suppress the taking of unnecessary checkpoints. However, the number of additional checkpoints is not minimized. Although this paper's algorithm uses integer vectors, the number of additional checkpoints is minimized. The author [6] has shown a distributed checkpoint algorithm which obtains a first and last consistent global checkpoint. Independent checkpointing algorithms, such as that in [9], do not obtain 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 for rollback.

2 Consistent 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 asynchronous; 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

[1] In order to roll back, the messages that 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 [5]. This paper thus discusses obtaining a consistent global checkpoint.

[2] The same algorithm is shown in [3] independently. However, they do not consider the following modification.

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 [8]. 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 [4].

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 sequence.

$\perp_i$ is an imaginary event which is p_i's initial state. For any p_i event e_i, $\perp_i \rightarrow e_i$ holds. This paper considers $\perp_i$ as checkpoints in E.

For p_i's event e_i in $E(A)$, causal-past event on p_j, $cp_i^{e_i}(j)$, is defined as follows.

Definition 2

- $cp_i^{e_i}(i) = e_i$.
- *$cp_i^{e_i}(j)$ is last event e_j in p_j that satisfies $e_j \rightarrow e_i$. If there is no event e_j satisfying $e_j \rightarrow e_i$, $cp_i^{e_i}(j) = \perp_j$.* ■

Intuitively, $cp_i^{e_i}(j)$ is p_j's last event which is known to p_i at e_i. In Fig. 1(c), $cp_1^{c_1^1}(1) = c_1^1$, $cp_1^{c_1^1}(2) = s(m_1)$, and $cp_1^{c_1^1}(3) = \perp_3$.

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.* ■

In Fig. 1(c), (c_1^1, c_2^1, c_3^1) is consistent, but (c_1^1, c_2^0, c_3^1) is not consistent because $c_2^0 \rightarrow c_1^1$.

A consistent global checkpoint for p_k's checkpoint initiation c_k in $E(A)$ is denoted as $gc(c_k, E(A))$. p_i's checkpoint in $gc(c_k, E(A))$ is denoted as $gc(c_k, E(A), i)$. $E(A)$ is omitted if it is obvious.

Theorem 1 *There is no algorithm that minimizes the total number of additional checkpoints in any execution.* ■

(Proof) Consider execution E in Fig. 1(a) and two algorithms, A and A'. A takes an additional checkpoint c_2^1 before $r(m_2)$. A' does not take the one. If execution E has no initiation other than c_3^1, A is worse than A' since $gc(c_3^1, E(A'))$ can be (c_1^0, c_2^0, c_3^1) and c_2^1 is unnecessary.

Next consider the execution E' in Fig. 1(b) in which another initiation, c_1^1, exists. Since $c_2^0 \rightarrow c_1^1$, A' must take $c_2^{1'}$ before $r(m_3)$ for $gc(c_1^1, E'(A'), 2)$. Since $c_3^1 \rightarrow c_2^{1'}$, p_3 must take c_3^2 before $r(m_4)$ for $gc(c_1^1, E'(A'), 3)$. Thus, $gc(c_1^1, E'(A')) = (c_1^1, c_2^{1'}, c_3^2)$ and the number of additional checkpoints of $E'(A')$ is 2.

For this execution, $gc(c_3^1, E'(A))$ can be (c_1^0, c_2^1, c_3^1) and $gc(c_1^1, E'(A))$ can be (c_1^1, c_2^1, c_3^1) as in Fig. 1(c) and the number of additional checkpoints of $E'(A)$ is 1. Thus A' is worse than A. ■

In Fig. 1(a), taking c_2^1 just before $r(m_2)$ reduces the number of additional checkpoints only if there is initiation c_1^1 which is unknown at $r(m_2)$. From the assumption, A cannot predict the future execution of a process. Thus, the following assumption is added, which means additional checkpoints are taken for "known" initiations.

Assumption 1 *Checkpointing algorithm A takes additional checkpoint c_i just before an event e_i only if $gc(c, E(A))$ cannot be obtained without c_i for initiation c satisfying $c \rightarrow e_i$.* ■

3 An algorithm to obtain consistent global checkpoints

Chandy and Lamport's distributed snapshot algorithm obtains one consistent global checkpoint for a set of concurrent initiations. This rule can be formally written as Rule 1 when processes continue to initiate checkpoints. A global checkpoint number (GCN) is assigned to each initiation. One consistent global checkpoint is obtained for each GCN.

Each process p_i has an integer vector $gcn_i(j)$. $gcn_i(j) = y$ means that p_i knows that p_j knows the maximum GCN is y. $gcn_i(i)$ is the current GCN p_i knows. GCN is assigned using gcn as follows:

(**Rule 1**: GCN assignment rule for process p_i)

(1) Initially, set $gcn_i(j) := 0$ for all j.

(2) When initiation c_i occurs, increment $gcn_i(i)$ and assign the number as the GCN for c_i.

(3) When p_i sends a message m, gcn_i's current value is piggybacked on m.

(4) When p_i receives a message m from p_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 $gcn_i(i) := max(gcn_i(i), mgcn(j))$. ■

Note that maintaining an integer rather than a vector is sufficient for GCN assignment. This vector is used to remove old checkpoints.

Lemma 1 *If initiations c_i and c_j have the same GCN, they are consistent.* ■

(Proof) Assume that $c_i \rightarrow c_j$ and GCN y is assigned to c_i. Because of Rule 1, $gcn_j(j) \geq y$ before c_j. Thus, y is not assigned to c_j. ■

The consistent global checkpoint for GCN y is denoted as $CGC(y)$. p_j's element of $CGC(y)$ is denoted as $CGC(y, j)$. Note that $CGC(0) = (\perp_1, \perp_2, \ldots, \perp_n)$.

In the rest of the paper, a sequence number is assigned for (both initiation and additional) checkpoints in each process. $\perp_i$ 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.

This algorithm uses the following variables. Variable $ck_i(j) = x(\geq 0)$ at p_i if $c_j^x \rightarrow e_i$ is satisfied, where e_i is p_i's current event. $ck_i(j) = -1$ if $\perp_j \not\rightarrow e_i$. $ck_i(i)$ is p_i's newest checkpoint number.

Variable $see_i(j) = true$ if p_i knows that there is a checkpoint c satisfying $c_j^x \rightarrow c$, where c_j^x is p_j's newest (the $ck_i(j)$-th) checkpoint. These variables are updated by a method similar to gcn update. The details are shown in Fig. 3.

Variable $st_i(j) = true$ if p_i sends a message to p_j after p_i's newest (the $ck_i(i)$-th) checkpoint. Variable $cgc_i(y)$ has the output of the algorithm. $cgc_i(y) = x_i$ if $CGC(y, i) = c_i^{x_i}$.

Consider the case when p_i receives a message m from p_j and $gcn_i(k)(k \neq i)$ and $see_i(k)$ are updated by the values on m. Let $y_0 = gcn_i(i)$ and $y_1 = mgcn(j)$. If $y_1 > y_0$, there is a checkpoint $c_j^{x_y}$ satisfying $CGC(y, j) = c_j^{x_y}$ and $c_j^{x_y} \rightarrow r(m)(y_0 < y \leq y_1)$, since p_j has decided $CGC(y, j)$. Since any p_i checkpoint after $r(m)$ is not consistent with $c_j^{x_y}$, $CGC(y, i)$ must be before $r(m)$. Thus, p_i must do one of the following: (1) take an additional checkpoint before $r(m)$ and set $cgc_i(y)$ as the new checkpoint, or (2) set $cgc_i(y)$ as an old checkpoint. The CGC decision rule is as follows:

(**Rule 2**: CGC decision rule for p_i)
Let $y_0 = gcn_i(i)$, $y_1 = mgcn(j)$ ($y_1 > y_0$), and $x_i = ck_i(i)$ at the arrival of message m from p_j. Take a checkpoint before $r(m)$ and set $cgc_i(y) := x_i + 1$ for all $y(y_0 < y \leq y_1)$ if condition (2-1) or (2-2) is satisfied (Fig. 2).
(2-1) $see_i(i) = true$.
(2-2) For some h, $st_i(h) = true$ and $gcn_i(h) < y_1$.
Otherwise, set $cgc_i(y) := x_i$ for all $y(y_0 < y \leq y_1)$. ■

The algorithm CGC based on Rules 1 and 2 is shown in Fig. 3. The correctness is shown below.

Lemma 2 *If a global checkpoint $(c_1, c_2, \ldots, c_n)$ 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.* ■

(Proof) Assume that the pair with the shortest causal sequence is (c_i, c_j) and that the length is more than 1. Let m_1 be the first message in the causal sequence and $p_k(\neq p_i, p_j)$ be the process that executes $r(m_1)$. If c_k is before $r(m_1)$, $c_k \rightarrow c_j$, and this contradicts the notion that (c_i, c_j) is the pair with the shortest causal sequence. If c_k is after $r(m_1)$, $c_i \rightarrow c_k$, and this also contradicts the minimality of the causal sequence. ■

Theorem 2 *The global checkpoints obtained by algorithm CGC are consistent.* ■

(Proof) Assume that $CGC(y)$ is not consistent. From lemma 2, assume that $CGC(y, i) \rightarrow CGC(y, k)$ and let the message in the causal sequence be m.

Let e_h^y be the event when p_h decides $CGC(y, h)$. e_h^y is an initiation or a receive event. If e_h^y is a receive event, $CGC(y, h)$ is before e_h^y. Otherwise, $e_h^y = CGC(y, h)$. Thus, $CGC(y, h)$ is p_h's newest checkpoint at e_h^y and $CGC(y, h) \rightarrow e_h^y$ is satisfied. $gcn_h(h) < y$ is satisfied before e_h^y and $gcn_h(h) \geq y$ is satisfied at e_h^y.

(**Case 1**: $e_k^y \rightarrow e_i^y$) Since $CGC(y, i)$ is p_i's newest checkpoint at e_i^y and $CGC(y, i) \rightarrow CGC(y, k) \rightarrow e_k^y \rightarrow e_i^y$, $see_i(i) = true$ at e_i^y. Thus, $CGC(y, i)$ must be the newly taken checkpoint just before e_i^y from Rule 2. This contradicts the fact that there is an event $s(m)$ between $CGC(y, i)$ and e_i^y.

(**Case 2**: e_i^y is before $s(m)$) Since $gcn_i(i) \geq y$ at e_i^y, $gcn_k(k) \geq y$ must be satisfied at $r(m)$. Thus, e_k^y must be equal to or before $r(m)$. This contradicts the notion that $CGC(y, k)$ is after $r(m)$.

(**Case 3**: $e_k^y \not\rightarrow e_i^y$ and e_i^y is after $s(m)$) Since $e_k^y \not\rightarrow e_i^y$, $gcn_i(k) < y$ at e_i^y. Since there is event $s(m)$ between $CGC(y, i)$ and e_i^y, e_i^y is a receive event from a process p_j. Let $y_1 = mgcn(j)$ at e_i^y. $y_1 \geq y$ is satisfied. Since $st_i(k) = true$ and $gcn_i(k) < y_1$ at e_i^y, $CGC(y, i)$ must be the newly taken checkpoint just before e_i^y from Rule 2. This contradicts the notion that there is event $s(m)$ between $CGC(y, i)$ and e_i^y. ■

Theorem 3 *When additional checkpoint $c_i^{x_i}$ is taken at e_i by Rule 2 under the GCN assignment rule Rule 1, there is an execution in which a consistent global checkpoint for an*

initiation c satisfying $c \rightarrow e_i$ cannot be obtained without $c_i^{x_i}$. ■

(Note) In Rule 2, $CGC(y,i)$ is set as the newest checkpoint before $r(m)$ for every $y(y_0 < y \leq y_1)$ when no additional checkpoint is taken. When an additional checkpoint is taken, $CGC(y,i)$ is set as the checkpoint for every $y(y_0 < y \leq y_1)$. In either case, a consistent global checkpoint might also be obtained by setting $CGC(y',i)$ as a checkpoint before the newest one for some $y'(y_0 < y' \leq y_1)$. This modification of Rule 2 does not reduce the total number of checkpoints. Since the following proof does not assume that $CGC(y',i)$ is set as the newest checkpoint, the additional checkpoints taken by Rule 2 are necessary no matter how $CGC(y',i)$ is selected.

(Proof) Assume that $r(m)$ in p_i is an event when an additional checkpoint is taken by Rule 2.

First consider the case when (2-1) is satisfied. Since $see_i(i) = true$, there is a checkpoint $c_k^{x_k}$ satisfying $c_i^{x_i} \rightarrow c_k^{x_k}$ and $c_k^{x_k} \rightarrow r(m)$. There is a consistent global checkpoint CGC_0 which includes $c_k^{x_k}$. If no checkpoint is taken at $r(m)$, $CGC_0(i)$ must be equal to or before $c_i^{x_i}$. In either case, CGC_0 is not consistent since $c_i^{x_i} \rightarrow c_k^{x_k}$.

Next consider the case when (2-1) is not satisfied but (2-2) is satisfied. Note that $gcn_i(k) \leq y_1$ for every k at $r(m)$. Let M be the first message sent to p_h after $c_i^{x_i}$. Let $y_2 = gcn_i(i)$ at $s(M)$. Since $s(M)$ is before $r(m)$, $y_2 \leq y_0 (< y_1)$ is satisfied.

Let $e_h = cp_i^{r(m)}(h)$, that is, p_h's last event known to p_i at $r(m)$. Let $y_3 = gcn_h(h)$ at e_h. Since $gcn_h(h)$ at e_h equals $gcn_i(h)$ at $r(m)$, $y_3 < y_1$ is satisfied.

If $y_2 < y_0$, there is a receive event $r(m')$ between $s(M)$ and $r(m)$ such that $gcn_i(i) = y_0$ after $r(m')$. Since p_i did not take a checkpoint at $r(m')$, $gcn_i(h) = y_0$ must be satisfied at $r(m')$. Thus, $y_0 \leq y_3$ holds. Therefore, $y_0 \leq max(y_2, y_3) < y_1$ is satisfied in every case.

Now consider the following execution after e_h. p_h executes $r(M)$ if $r(M)$ is not before e_h. p_h then initiates checkpoint c_h^x. Let the GCN for c_h^x be y_4. Since $gcn_h(h)$ just before c_h^x is $max(y_2, y_3)$, $y_4 = max(y_2, y_3) + 1 \leq y_1$ is satisfied. If no checkpoint is taken at $r(m)$, $CGC(y_4, i)$ is equal to or before $c_i^{x_i}$ and $CGC(y_4)$ is not consistent because $c_i^{x_i} \rightarrow c_h^x$. ■

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

Here, the rule for removing old checkpoints is shown. The amount of stable storage usage becomes large if old checkpoints are not removed. When p_i has a failure, the processes use the consistent global checkpoint that includes p_i's newest checkpoint. Let $y_0 = min_j gcn_i(j)$. p_i knows that no process uses $CGC(y)(y < y_0)$ for rollback, thus p_i can remove the checkpoints before $cgc_i(y_0)$.

4 Modification for inconsistent initiations

Rule 1 forces taking a different consistent global checkpoint for two initiations c, c' satisfying $c \rightarrow c'$. However, there are cases when it is unnecessary to obtain a consistent global checkpoint for c'. This section modifies Rule 1 to suppress the need for taking additional checkpoints.

For a consistent global checkpoint CGC and a set of pairs of a process number and checkpoint $S = \{(i_1, c_{i_1}^{x_{i_1}}), (i_2, c_{i_2}^{x_{i_2}}), \ldots, (i_k, c_{i_k}^{x_{i_k}})\}$, denote the global checkpoint defined below as $GC(CGC; S)$:

$$GC(CGC; S, i) = \begin{cases} c_i^{x_i} & \text{if } (i, c_i^{x_i}) \in S \\ CGC(i) & \text{if } (i, *) \notin S \end{cases}$$

If $GC(CGC; S)$ is consistent, the procedure for obtaining a consistent global checkpoint that includes S can be stopped halfway.

First, consider the case when $|S| = 1$. No new GCN is assigned to initiation $c_i^{x_i}$ if $GC(CGC^*; \{(i, c_i^{x_i})\})$ is consistent, where CGC^* is the consistent global checkpoint which includes $c_i^{x_i - 1}$. Rule 1 (2) is changed as follows. Note that (1),(3), and (4) are unchanged.

(**Rule 1'**: modified GCN assignment rule for p_i)

(2) When initiation $c_i^{x_i}$ occurs, increment $gcn_i(i)$ and assign the number as the GCN for $c_i^{x_i}$ if one of the following conditions is satisfied.

(1'-1) $gcn_i(j) = gcn_i(i)$ for some $j (\neq i)$.

(1'-2) There is a process p_j satisfying (a)-(d).

(a) p_j takes a checkpoint $c_j^{x_j}$ satisfying $c_j^{x_j} \not\rightarrow c_i^{x_i - 1}$ and then sends a message m to p_i.

(b) p_i executes $r(m)$ between $c_i^{x_i - 1}$ and $c_i^{x_i}$.

(c) There is no message M such that M is sent from p_j after $c_j^{x_j}$ to a process other than p_i and $r(M) \rightarrow c_i^{x_i}$.

(d) At least one message m' is sent from p_i to process $p_k (\neq p_j)$ after $gcn_i(i)$ becomes the current value, if p_j sends message m_0 to a process other than p_i satisfying $c_j^{x_j} \rightarrow s(m_0) \rightarrow c_i^{x_i}$. If p_j does not send such message m_0, there is no restriction on the receiver of m'. ■

Theorem 4 *$GC(CGC^*; \{(i, c_i^{x_i})\})$ is consistent if CGC^* is consistent and the conditions in Rule 1' are not satisfied for $c_i^{x_i}$.* ■

(Proof) Assume that CGC^* is consistent and $GC(CGC^*; \{(i, c_i^{x_i})\})$ is not consistent.

Let $y = gcn_i(i)$ at $c_i^{x_i - 1}$. $CGC^*(j) = CGC(y, j)$ for all $j (\neq i)$. $CGC(y, j) \not\rightarrow c_i^{x_i - 1}$ for all $j (\neq i)$, since CGC^* is consistent.

If $c_i^{x_i} \rightarrow CGC(y,j)(j \neq i)$, $c_i^{x_i-1} \rightarrow CGC(y,j)$ and CGC^* is not consistent. This case cannot occur. Thus, $CGC(y,j) \rightarrow c_i^{x_i}$ holds for some $j(\neq i)$. Let e_j^y be the event when p_j decides $CGC(y,j)$. Without loss of generality, assume that j satisfies $e_h^y \not\rightarrow e_j^y$ for any $h(\neq j)$ satisfying $CGC(y,h) \rightarrow c_i^{x_i}$. If e_j^y satisfies $e_j^y \rightarrow c_i^{x_i}$, $gcn_i(j) = y$ at $c_i^{x_i}$ and (1'-1) is satisfied.

Next consider the case $e_j^y \not\rightarrow c_i^{x_i}$. Let a longest causal sequence from $CGC(y,j)$ to $c_i^{x_i}$ be $CGC(y,j), s(m_1), \ldots, r(m_l), c_i^{x_i}$. If $r(m_l)$ is before $c_i^{x_i-1}$, CGC^* is not consistent. Thus, $r(m_l)$ is after $c_i^{x_i-1}$. Since $e_j^y \not\rightarrow c_i^{x_i}$, e_j^y is after $s(m_1)$. $l = 1$ is shown. Assume that $l \geq 2$. Let the process that executes $r(m_1)$ be p_k. If $CGC(y,k)$ is after $r(m_1)$, CGC^* is not consistent because $CGC(y,j) \rightarrow CGC(y,k)$. Thus, $CGC(y,k)$ is before $r(m_1)$ and $CGC(y,k) \rightarrow c_i^{x_i}$. $e_k^y \not\rightarrow e_j^y$ is satisfied from the definition of j. Since $e_k^y \not\rightarrow e_j^y$, $gcn_j(k) < y$ at e_j^y. Since $st_j(k) = true$ at e_j^y, $CGC(y,j)$ cannot be an old checkpoint before $s(m_1)$ from Rule 2. Therefore, $l = 1$ holds and $CGC(y,j)$ satisfies (a) and (b) in (1'-2).

Assume that there is a message M sent to a process other than p_i after $CGC(y,j)$ and $r(M) \rightarrow c_i^{x_i}$. Thus there is a causal sequence $CGC(y,j), s(M), r(M), \ldots, c_i^{x_i}$ whose length is more than one. This contradicts the notion that the maximum length of causal sequences from $CGC(y,j)$ to $c_i^{x_i}$ is 1. Thus, such a message does not exist. Therefore, (c) in (1'-2) is satisfied.

$st_j(i) = true$ at e_j^y. Thus, $gcn_j(i) \geq y$ must be satisfied at e_j^y in order to select an old checkpoint as $CGC(y,j)$ at e_j^y. Therefore, $e_i^y \rightarrow e_j^y$ must be satisfied. If $c_i^{x_i} \rightarrow e_j^y$, $see_j(j) = true$ at e_j^y and $CGC(y,j)$ must be the newly taken checkpoint just before e_j^y. This contradicts the notion that there is an event $s(m_1)$ between $CGC(y,j)$ and e_j^y. Thus, $c_i^{x_i} \not\rightarrow e_j^y$ and p_i must have sent a message m' between e_i^y and $c_i^{x_i}$.

In addition, if $st_j(k) = true$ at $cp_i^{x_i}(j)$ for some $k(\neq i)$, $gcn_j(k) \geq y$ must also be satisfied at e_j^y. If ($gcn_j(k) < y$ and $gcn_j(i) \geq y$) or ($gcn_j(k) \geq y$ and $gcn_j(i) < y$) at some event e_j, p_j takes an additional checkpoint just before e_j and this contradicts the existence of $s(m_1)$ between $CGC(y,j)$ and e_j^y. If m' is sent to p_j and no other message is sent from p_i between e_i^y and $c_i^{x_i}$, one of the above conditions is satisfied at $r(m')$ or just before $r(m')$ because $gcn_i(k) < y$ at $s(m')$. Thus, p_i must have sent a message to a process other than p_j between e_i^y and $c_i^{x_i}$. Therefore (d) in (1'-2) is satisfied. ■

Theorem 5 *When $c_i^{x_i}$ satisfies one of the conditions of Rule 1', there is an execution in which $GC(CGC^*; \{(i, c_i^{x_i})\})$ is not consistent.* ■

(Proof) Let $y = gcn_i(i)$ at $c_i^{x_i-1}$. First, consider the case when (1'-1) is satisfied. In this case, $CGC(y,j) \rightarrow c_i^{x_i}$ and $GC(CGC^*; \{(i, c_i^{x_i})\})$ is not consistent.

Next, consider the case when (1'-1) is not satisfied and (1'-2) is satisfied. $gcn_h(h) < y$ is satisfied at $cp_i^{x_i}(h)$ for any $h(\neq i)$ because (1'-1) is not satisfied. Let the message sent from p_i to p_k be m'. Note that $r(m')$ is after $cp_i^{x_i}(k)$. Otherwise, $gcn_k(k) = y$ at $cp_i^{x_i}(k)$. Let the set of processes to which p_j sends a message after $c_j^{x_j}$ other than p_i be $P_j = \{p_{j_1}, p_{j_2}, \ldots, p_{j_l}\}$. For any message M sent from p_j to p_{j_h} after $c_j^{x_j}$, $r(M)$ is not before $cp_i^{x_i}(j_h)$. Otherwise, $r(M) \rightarrow c_i^{x_i}$, and this contradicts (1'-2)(c). Consider the following execution after $cp_i^{x_i}$. Assume that $P_j \neq \phi$. From (d), $p_k \neq p_j$. Consider the case $p_k \notin P_j$. When $p_k \in P_j$ or $P_j = \phi$, similar executions can be constructed. The details are omitted here.

p_k executes $r(m')$ and sends M_0 to p_{j_1} just after $cp_i^{x_i}(k)$. $p_{j_h}(h = 1, \ldots, l-1)$ executes $r(M_{h-1})$ and sends M_h to $p_{j_{h+1}}$ just after $cp_i^{x_i}(j_h)$, that is, before receiving any message from p_j. p_{j_l} executes $r(M_{l-1})$ and sends M_l to p_j just after $cp_i^{x_i}(j_l)$. p_j executes $r(M_l)$. $see_j(j) = false$ is satisfied at $r(M_l)$. Assume that $see_j(j) = true$. There is a checkpoint c satisfying $c_j^{x_j} \rightarrow c$ and $c \rightarrow r(M_l)$. c must not be in p_i because there is no checkpoint between $c_i^{x_i-1}$ and $c_i^{x_i}$. If c is in $p_h(h \neq i)$, c must satisfy $r(m') \rightarrow c$ in order to satisfy $c_j^{x_j} \rightarrow c$. However, there is no such checkpoint in this execution. Thus, $see_j(j) = false$ at $r(M_l)$.

In addition, $gcn_j(h) = y$ for any process p_h such that $st_j(h) = true$ at $r(M_l)$. Thus, p_j sets $c_j^{x_j}$ as $CGC(y,j)$ and $GC(CGC^*; \{(i, c_i^{x_i})\})$ is not consistent since $c_j^{x_j} \rightarrow c_i^{x_i}$. ■

Theorem 6 *When additional checkpoint $c_i^{x_i}$ is taken at e_i by Rule 2 under the GCN assignment rule Rule 1', there is an execution in which a consistent global checkpoint for an initiation c satisfying $c \rightarrow e_i$ cannot be obtained without $c_i^{x_i}$.* ■

The proof is similar to the one for Theorem 3 and omitted [5].

Lastly, the case $|S| \geq 2$ is discussed. Even if Rule 1' cannot be applied and GCN $y+1$ is assigned to the initiations in S, obtaining $CGC(y+1)$ seems to be able to be stopped halfway if $GC(CGC(y); S)$ is consistent. Actually, this is impossible and the reason is shown below.

It is obvious that $|S| = n-1$ is the case of selecting a checkpoint in $CGC(y)$ for the last element of $CGC(y+1)$, and this case has been discussed in Rule 2. Thus assume that $2 \leq |S| \leq n-2$ and there are two processes, p_h and p_k, satisfying $(h, *), (k, *) \notin S$. Consistent $GC(CGC(y); S)$ can be obtained if each element in $GC(CGC(y); S) - S$ can be selected as $CGC(y+1)$ by Rule 2. Thus, it is assumed that at p_k's (p_h's) receive event $r(m_k)$ ($r(m_h)$), which informs GCN $y+1$, Rule 2 forces p_k (p_h) to take an additional checkpoint before the receive event and set it as $CGC(y+1, k)$ ($CGC(y+1, h)$). With these assumptions, the following theorem holds.

Theorem 7 *There is no algorithm in which p_h or p_k do not take an additional checkpoint at $r(m_k)$ and $r(m_h)$ when $GC(CGC(y); S)$ is consistent.* ■

(Sketch of the proof) It is necessary for p_h and p_k to have common knowledge [2] that $GC(CGC(y); S)$ is consistent. However, common knowledge cannot be attained when the communication is asynchronous [2]. ■

5 Conclusion

This paper showed a distributed algorithm which obtains a consistent global checkpoint for any initiation. Among the extensions of Chandy and Lamport's algorithm, this algorithm minimizes the number of additional checkpoint. One open question is whether this algorithm is optimal without Chandy and Lamport's assumption.

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

References

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

[2] Halpern, J. Y. and Moses, Y.: "Knowledge and Common Knowledge in a Distributed Environment," Journal of the ACM, Vol. 37, No. 3, pp. 549–587 (July 1990).

[3] 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).

[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).

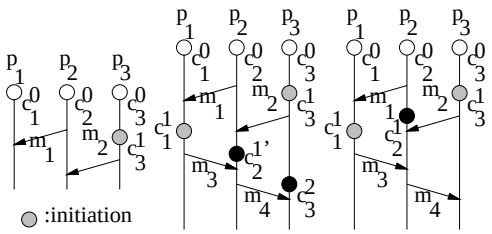
[5] Manabe, Y.: "A Distributed Consistent Global Checkpoint Algorithm with a Minimum Number of Checkpoints," Technical Report of IEICE, COMP97-6 (Apr. 1997).

[6] Manabe, Y.: "A Distributed First and Last Consistent Global Checkpoint Algorithm," Proc. of 12th Int. Conf. on Information Networking (Jan. 1998).

[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] 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).

[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).



(a) Execution E. (b) $E'(A')$. (c) $E'(A)$.

Figure 1. Minimizing additional checkpoints.

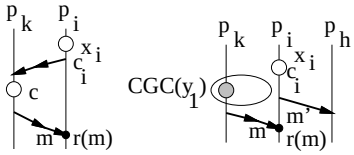


Figure 2. Rule 2-1 and 2-2.

```
program CGC; /* program for p_i. */
const n = ...; /* number of processes */
var gcn(n), ck(n), cgc(*): integer;
  st(n), see(n): boolean;
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;
end; /* end of subroutine */
/* main */
initialization begin
  for each k do gcn(k) := 0;
  for each k(≠ i) do ck(k) := −1;
  ck(i) := 0;
  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;
  gcn(i) := gcn(i) + 1;
  cgc(gcn(i)) := ck(i);
end; /* end of checkpoint initiation */
when p_i sends m to p_j begin
  send(m, gcn, ck, see) to p_j;
  st(j) :=true;
end; /* end of message sending */
when p_i receives (m, mgcn, mck, msee) from p_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));
  if gcn(i) < mgcn(j) then begin
    if see(i) or (∃h, st(h) and gcn(h) < mgcn(j))
      then checkpoint;
    for each y(gcn(i) < y ≤ mgcn(j)) do cgc(y) := ck(i);
  end;
  gcn(i) := max(gcn(i), mgcn(j));
  execute r(m);
end; /* end of message receiving */
```

Figure 3. Algorithm CGC.