A Consistent Global Checkpoint Algorithm for Distributed Systems with a Forbidden Process

Yoshifumi Manabe
NTT Basic Research Laboratories
3-1 Morinosato-Wakamiya, Atsugi-shi, Kanagawa 243-0198 Japan
manabe@theory.brl.ntt.co.jp

Abstract

A distributed coordinated checkpointing algorithm for distributed systems with a special process, called a forbidden process, is discussed. A consistent global checkpoint is a set of states in which no message is recorded as received in one process and as not yet sent in another process. It is used for rollback when a process failure occurs. The number of checkpoints in the forbidden process must be minimized because of its heavy load or its low stable storage capacity. A distributed checkpointing algorithm which takes the minimum number of checkpoints in the forbidden process is presented.

1 Introduction

Distributed coordinated checkpointing is a fundamental method to recover distributed systems after failure [3]. It 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. When there is process failure, execution can be continued from the set of rolled-back states if every process rolls back to each state in a consistent global checkpoint and the messages that have been sent and not received are restored. The message restoration method is similar to Strom and Yemini’s [10] and the details are discussed in [7]. This paper discusses how to obtain a consistent global checkpoint.

When a process initiates checkpointing, it takes its checkpoint and notifies the other processes about the initiation. Each of the other processes receives this information, it might have to take its checkpoint to obtain a consistent global checkpoint. Throughout this paper, the checkpoint taken by the initiator is called an initiation. The other checkpoints are called additional checkpoints.

All former checkpoint algorithms considered the system to be symmetric, i.e., that the disadvantage of taking an additional checkpoint in every process would be the same. This paper considers a case where the distributed system contains a special process in which the number of additional checkpoints must be minimized. We call such a process a forbidden process. For example, consider a system which consists of a huge server and many small size clients. If the server has a very large database and offers important services, the services it offers might stop for a long time when the server takes a checkpoint. The clients’ checkpointing might finish quickly and its influence might be relatively small. Thus, taking fewer checkpoints in the server and more checkpoints in the clients would be better than the usual checkpointing. Such a server can be the forbidden process. Another example is a system with a mobile process. Since the stable storage capacity is very low in the mobile process, many checkpoints cannot be taken in the mobile process. Here, the mobile process can be the forbidden process. This paper describes a checkpointing algorithm in which the number of additional checkpoints in the forbidden process is minimized.

2 Consistent global checkpoint

A distributed system is modeled by a finite set of processes \( \{p_1, p_2, \ldots, p_n\} \) interconnected by point-to-point channels. \( p_1 \) is the forbidden process and \( p_j (j \neq 1) \) are the other processes called normal processes.

Channels are assumed to be error-free and to have infinite capacity. The communication is asynchronous; that is, the delay experienced by a message is unbounded but finite. Channels might not be FIFO (First-In, First-Out).

\( p_i \)'s execution is a sequence of \( p_i \)'s events which include checkpoint initiations. System execution \( E \) is the set of each process’s execution. \( p_i \)'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 “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 \( e' \).
Effective in non-FIFO channels [6].

Figure 1: System execution.

(2) e is the send event s(m) and e' is the receive event r(m) of the same message m.
(3) e → e'' and e'' → e' for event e'”.

Two special events, ⊥, and T, are defined for p. ⊥ is an imaginary event which is p’s initial state. T is p’s current event if p is not terminated. If p is terminated, T is an imaginary event which is p’s terminal state. For any p event e, ⊥ → e and e → T. This paper considers ⊥ and T as checkpoints in E.

Definition 2 A pair of checkpoints (c, c’) is consistent if and only if c ≠ c’ and e’ ≠ c.

A global checkpoint (c1, c2, . . . , cn) is an n-tuple of checkpoints where ci is pi’s checkpoint. A global checkpoint is consistent if and only if all distinct pairs of checkpoints are consistent.

In Fig. 1, (c1, c2, c3) is consistent, but (c1, c2, c3) is not consistent because of message m3.

A consistent global checkpoint for pk’s checkpoint initiation ck in E(A) is denoted as gc(ck, E(A)). pk’s checkpoint in gc(ck, E(A)) is denoted as gc(ck, E(A), i). E(A) is omitted if it is obvious.

Checkpointing algorithm can be classified into two groups. The former uses special messages called markers [1]. The latter is communication-induced algorithm [2][5][6][9], in which all information for checkpointing is piggybacked on massages in E. This paper discusses the latter algorithm, since markers are not effective in non-FIFO channels [6].

3 The checkpointing algorithm

In the rest of the paper, a sequence number is assigned for the (both initiation and additional) checkpoints in each process. ⊥, is p’s 0-th checkpoint. Let c*k be pk’s xk-th checkpoint.

Let the newest checkpoint for the forbidden process p1 be c1*. p1 must take an additional checkpoint before r(m) if the following condition is satisfied: there is an initiation c which satisfies c* → c and c → r(m) (Fig. 2). Suppose that p1 does not take an additional checkpoint just before r(m). If gc(c, E(A), 1) is c1* or before c1*, gc(c, E(A), 1) → c and gc(c, E(A)) is not consistent. If gc(c, E(A), 1) is a checkpoint after r(m), c → gc(c, E(A), 1) and gc(c, E(A)) is not consistent.

Theorem 1 Any algorithm A must take an additional checkpoint just before r(m) in p1 if there is an initiation c such that c* → c and c → r(m), where c* is p1’s newest checkpoint.

If this rule is the only one rule that forces p1 to take an additional checkpoint, the number of p1’s additional checkpoints is minimized.

The algorithm in [2][6] for systems without forbidden processes assigns a global checkpoint number (GCN) to each initiation. GCN update rule is similar to the one of Lamport’s logical clock [4]. If the algorithm in [6] is applied to the execution in Fig. 1, GCN for c1, c2, c3, and c4 are 1, 2, 1, 2, and 3, respectively. Initiations with the same GCN are consistent and one consistent global checkpoint is obtained for each GCN.

This paper uses the above rule for normal processes. Note that the condition “obtain one consistent global checkpoint for one GCN” in [2][6] does not minimize the number of additional checkpoints. I [6] modified the rule to reduce the number of additional checkpoints. However, the modified rule is complicated and the optimality of the modified rule is an open question. This paper thus uses the above basic rule to simplify.
the algorithm. Therefore, the number of additional checkpoints in normal processes is not minimized by this algorithm.

New numbering for systems with a forbidden process, the modified global checkpoint number MGCN, consists of two integers \((xgcn, ygcn)\). \(xgcn\) shows the number of checkpoints in \(p_1\). \(ygcn\) is just the same as the GCN among \(\{p_2, \ldots, p_n\}\). \(ygcn\) is reset to 0 when a new \(xgcn\) arrives. Formally, MGCN setting rule is as follows. MGCN for \(p_1\)'s checkpoint \(c^1_i\) is \((x_0, 0)\), MGCN for \(\bot(i \neq 1)\) is \((-1, 1)\). MGCN for \(p_i(i \neq 1)\)'s initiation \(c^i_\text{ini}\) is \((x_0, y_0)\), where \(x_0 = \max_x\{c^i_x | c^i_x \rightarrow c^i_f\}\) and \(y_0 = 1 + \max_y\{\text{MGCN} (x_0, y)\text{ for checkpoint } c | c \rightarrow c^i_f\}\). If \(c^i_f \notightarrow c^i_i\), \(x_0 = -1\). In Fig. 1, MGCN for \(c^1, c^2, c^3, c^4\) and \(c^5\) are \((1, 0), (2, 0), (0, 1), (1, 1), \) and \((1, 2)\), respectively.

For two MGCNs \((x, y), (x', y')\), \((x, y) > (x', y')\) if \(x > x'\) or \((x = x' \text{ and } y > y')\). \((x, y) \geq (x', y')\) if \((x, y) > (x', y')\) or \((x, y) = (x', y')\).

\(p_i(i \neq 1)\) assigns one checkpoint for each MGCN \((x, y)\). Let \(CA_i(x, y)\) be the checkpoint assigned to MGCN \((x, y)\) by \(p_i\). The same checkpoint might be assigned to multiple MGCNs. Suppose that \(p_i\)'s current MGCN is \((x_0, y_0)\) and \(p_i\) receives a message \(m\) from \(p_j\) which informs MGCN \((x_1, y_1)\) such that \((x_1, y_1) > (x_0, y_0)\). If \(x_1 > x_0\), \(CA_i(x_0, y_0)(y > y_0)\) will no longer be obtained. Let \(mxy_i(x)\) be the value of \(y\) in MGCN \((x, y)\) when new MGCN information such that \(x > x\) arrives at \(p_i\). If there is no initiation \((x, y)(y > 0)\) in \(p_i\), \(mxy_i(x) = 0\). \(mxy_i(x) = 0\) for any \(x\). In Fig. 1, \(mxy_2(1-1) = 1, mxy_3(1-1) = 1, \) and \(mxy_3(0) = 0\). For \(p_i\)'s initiation \(c^i_i\) whose MGCN is \((x, 0)\), the global checkpoint which contains \(c^i_j\), \(CGCx(x, i)\), is as follows: \(p_i\)'s element in \(CGCx(x, i)\), \(CGCy(x, y, i)\), is as follows: \(p_i\)'s element in \(CGCy(x, y)\), \(CGCy(x, y, i)\), is as follows: \(p_i\)'s element in \(CGCy(x, y, i)\), is as follows:

\[CGCy(x, y, i) = \begin{cases} CA_i(x, y) & \text{if } y \leq mxy_i(x) \\
CA_i(x + 1, 0) & \text{if } y > mxy_i(x) \end{cases} \]

In Fig. 1, \(CGCy(0, 0, 1) = CA_1(1, 0)\) and \(CGCy(0, 1, 3) = CA_2(1, 0)\).

The algorithm which assigns a checkpoint for each MGCN uses the following variables. \(p_i\)'s variable \(ck_i(j)\) has the number of checkpoints. \(ck_i(j) = x(\geq 0)\) if \(c^j_x \rightarrow c_i\) is satisfied, where \(c_i\) is \(p_i\)'s current event. \(ck_i(j) = -1\) if \(\bot \notightarrow c_i\), \(ck_i(i)\) is \(p_i\)'s newest checkpoint number. In Fig. 1, \(ck_3(1) = 1, ck_3(2) = 1, \) and \(ck_3(3) = 2\) at \(c^3_2\) in \(p_3\).

The boolean variable \(sec_j\) has information about the new checkpoint. \(sec_j(x) = true\) if \(p_i\) knows that there is a checkpoint \(c\) satisfying \(c^i_x \rightarrow c\) where \(c^i_x\) is \(p_i\)'s newest (the \(ck_i(j)\)-th) checkpoint. If there is no such checkpoint \(c\), \(sec_j(x) = false\). If \(sec_j(x) = true\) at \(p_i\)'s event \(c\), the following condition is satisfied. For \(p_i\)'s newest checkpoint \(c^i_3\), there is a checkpoint \(c\) which satisfies \(c^i_x \rightarrow c\) and \(c \rightarrow c\).

Variable \(xgcn_i(j)\) and \(ygcn_i(j)\) has information about MGCN. \((xgcn_i(j), ygcn_i(j)) = (x, y)\) if \(p_i\) is aware that \(p_j\) currently knows that maximum MGCN is \((x, y)\), that is, \((xgcn_j(j), ygcn_j(j)) = (x, y)\) at \(p_j\)'s event currently known to \(p_i\). \((xgcn_i(i), ygcn_i(i))\) is \(p_i\)'s current knowledge about maximum MGCN.

Variable \(st_i(j)\) has information about message sending. \(st_i(j) = true\) if \(p_i\) sends a message to \(p_j\) after \(p_i\)'s newest (the \(ck_i(i)\)-th) checkpoint.

Variable \(ca_i(x, y)\) has \(CA_i(x, y)\). Note that information about multiple MGCNs arrives at \(p_i\) at the same time. Suppose that current \((xgcn_i(i), ygcn_i(i)) = (x_0, y_0)\) and new information about MGCN \((x_1, y_1)\) arrives at \(r(m)\). This algorithm selects the same checkpoint \(c^i_x\) for every \(CA_i(x, y)\) such that \((x_0, y_0) < (x, y) \leq (x_1, y_1)\). In this case, the algorithm just sets \(ca_i(x_1, y_1) = x_1\). For a tuple \((x, y)\), \(CA_i(x, y)\) is obtained from \(ca_i\) as follows. Find the smallest pair \((x', y')\) which satisfies \((x', y') \geq (x, y)\) and \(ca_i(x', y')\) is defined. If \(ca_i(x', y') = x_1, CA_i(x, y) = c^i_x\). Such a pair \((x', y')\) is unique because the tuple \((xgcn_i(i), ygcn_i(i))\) never decreases in \(p_i\). A pair \((x', y')\) cannot be found only if \((xgcn_i(i), ygcn_i(i)) < (x, y)\), that is, the initiation \(c^i_x\) whose MGCN is \((x, y)\) is unknown in \(p_i\)'s current state; \((x,y)\) is considered to be \(\bot\) in the state.

The values of \(ck, sec, xgcn,\) and \(ygcn\) are updated by sending the current value on every message. It is shown in Fig. 5.

Now consider the case when \(p_i\) receives a message \(m\) from \(p_j\) and \(xgcn_i(k), ygcn_i(k)(k \neq i)\), and \(sec_i(k)\) are updated by the values on \(m\). Let \((x_0, y_0) = (xgcn_i(i), ygcn_i(i))\) and \((x_1, y_1)\) be the value of \((xgcn_i, ygcn_i)\) arrived on message \(m\) and \((x_1, y_1) > (x_0, y_0)\). \(p_i\) must assign a checkpoint for this new MGCN.

First consider the case when \(p_i\) is the forbidden process \(p_1\). Since no process other than \(p_1\) increments \(xgcn, x_1 = x_0\) is satisfied. There is an initiation \(c\) whose MGCN is \((x_0, y_1)\), where \(y_1 > 0\). \(c\) satisfies \(c^0_x \rightarrow c\) because of its \(xgcn\). Thus, \(c\) satisfies the condition in Theorem 1 because \(c \rightarrow r(m)\). Therefore, \(p_i\) needs to take a checkpoint whenever \((x_1, y_1) > (x_0, y_0)\) is satisfied.

Next consider the case when \(p_i\) is a normal process. A consistent global checkpoint which includes \(c\) whose MGCN is \((x_1, y_1)\) must be obtained. Since any \(p_i\) checkpoint after \(r(m)\) is not consistent with \(c, gc(c, i)\) must be before \(r(m)\). Thus, \(p_i\) must do one of the following: (1) take an additional checkpoint just before \(r(m)\) and set \(ca_i(x_1, y_1)\) as the new checkpoint, or (2) set \(ca_i(x_1, y_1)\) as an old checkpoint. In the second case, \(ca_i(x_1, y_1)\) is set to \(p_i\)'s newest checkpoint because of simplicity.

Let the newest checkpoint in \(p_i\) be \(c^i_x\). There are two cases to take an additional checkpoint before \(r(m)\). The first case is when there is a checkpoint \(c\) which
satisfies $c_i^x \rightarrow c$ and $c \rightarrow r(m)$ (Fig. 3 (a)). There can be an initiation $c'$ which satisfies $gc(c', k) = c$. $gc(c', i)$ must not be before $c_i^x$ because $c_i^x \rightarrow c = gc(c', k)$. Thus, without taking an additional checkpoint, $gc(c')$ cannot be consistent. This rule can be written as follows.

(Rule 1) $sec_i(i) = true$. The second case is when $p_i$ sends a message $m'$ after current checkpoint $c_i^x$ to process $p_h$ which satisfies $(xgcn_i(h), ygcn_i(h)) < (x_1, y_1)$ (Fig. 3 (b)). Assume that $p_h$ does not take an additional checkpoint just before $r(m)$. Here, $CA_i(x_1, y_1) = c_i^x$.

Figure 3: (a) Rule 1. (b) Rule 2.

First consider the case when $p_h$ is a normal process. The execution after current event might be as follows. $p_h$ executes $r(m')$. Note that $(xgcn_h(h), ygcn_h(h)) < (x_1, y_1)$ at $r(m')$. $p_h$ then receives information about MGCN $(x_1, y)(y < y_1)$ from another process if $xgcn_h(h) < x_1$. $p_h$ then initiates several times. Then, there might be an initiation $c_h$ whose MGCN is $(x_1, y_1)$. Then, $CGCy(x_1, y_1)$ is not consistent since $CA_i(x_1, y_1) \rightarrow c_h$.

Next consider the case when $p_h$ is the forbidden process $p_1$. Since $c_1^x$ is the newest checkpoint in $p_1$, $xgcn_1(1) = x_1$ is satisfied. Thus, $ygcn_1(1) < y_1$ holds. $CGCy(x_1, y_1, 1) = c_1^{x+1}$. The execution after current event might be as follows. $p_1$ executes $r(m')$. There can be cases when $xgcn_1(1) = x_1$ at $r(m')$. $p_1$ then initiate a checkpoint $c_1^{x+1}$ whose MGCN is $(x_1 + 1, 0)$. Then, $CGCy(x_1, y_1)$ is not consistent since $CA_i(x_1, y_1) \rightarrow e_1^{x+1}$.

This rule can be written as follows.

(Rule 2) $\exists h, (x_1, y_1) > (xgcn_i(h), ygcn_i(h))$ and $st_i(h) = true$.

The algorithm MGC which includes variable updating is in Fig. 5.

The correctness of the algorithm is proved below.

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

(Proof) First assume that $CGCx(x)$ is not consistent and there is an orphan message $m$ sent after $c_i^x (= CGCy(x, i))$ and received before $c_k^x (= CGCx(x, k))$.

Let $e_k^x$ be the event when $p_h$ decides $CGCx(x, h)$, that is, $ca_h(x', y')$ is set to $e_k^x$ for $(x', y')$ where $x' \geq x$. $e_k^x$ is an initiation, a receive event, or $\top_h$ (in this case, $CGCx(x, h) = \top_h$). If $e_k^x$ is a receive event, $c_k^h$ is before $e_k^x$. Otherwise, $c_k^h$ is $p_h$'s newest checkpoint at $e_k^x$ and $c_k^h \rightarrow e_k^x$ is satisfied. $xgcn_h(h) < x$ is satisfied before $e_k^x$ and $xgcn_i(h) \geq x$ is satisfied at $e_k^x$. Note that for the forbidden process $p_1$, $e_k^x = e_i^x$.

(Case 1) $e_k^x$ is before $s(m))$ Since $xgcn_i(i) \geq x$ at $e_i^x$, $xgcn_k(k) \geq x$ must be satisfied at $r(m)$. Thus, $e_k^x$ must be equal or be before $r(m)$. This contradicts the notion that $e_k^x$ is after $r(m)$.

(Case 2) $e_k^x$ is after $s(m))$ Since $e_i^x = e_f^x$, $p_i$ is not the forbidden process.

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

(Case 2-2) $e_k^x \neq e_i^x$ Since $e_k^x \neq e_i^x$, $xgcn_k(k) < x$ at $e_i^x$. Since there is event $s(m)$ between $c_i^x$ and $e_i^x$, $e_i^x$ is a receive event from process $p_1$. Let $x' = xgcn_i(j)$ at $e_i^x$. $x' \geq x$ is satisfied. Since $st_i(h) = true$ and $xgcn_i(k) < x' < e_i^x$, $e_i^x$ must be the newly taken checkpoint just before $e_i^x$ from Rule 2. This contradicts the notion that there is an event $s(m)$ between $c_i^x$ and $e_i^x$. Therefore, $CGCx(x)$ is consistent.

Next assume that $CGCy(x, y, i)(y > 0)$ is not consistent and there is an orphan message $m$ sent after $c_i^x (= CGCy(x, y, i))$ and received before $e_k^x (= CGCy(x, y, k))$.

Let $e_h^{xy}$ be the event when $p_h$ decides $CGCy(x, y, h)$. For the forbidden process $p_1$, $e_h^{xy}$ is $e_1^{x+1}$. For a normal process, $e_h^{xy}$ is the event when $ca_h(x', y')$ is set to $e_h^{xy}$ for $(x', y')$ where $(x', y') \geq (x, y)$. $e_h^{xy}$ is $p_h$'s newest checkpoint at $e_h^{xy}$ and $e_h^{xy} \rightarrow e_h^{xy}$ is satisfied. $(xgcn_h(h), ygcn_i(h)) < (x, y)$ is satisfied before $e_h^{xy}$. $(xgcn_h(h), ygcn_i(h)) \geq (x, y)$ is satisfied at $e_h^{xy}$.

(Case 1) $e_h^{xy}$ is before $s(m))$ First assume that $p_k$ is a normal process. Since $(xgcn_i(i), ygcn_i(i)) > (x, y)$ at $e_i^{xy}$, $(xgcn_k(k), ygcn_i(k)) > (x, y)$ must be satisfied
at \( r(m) \). Thus, \( e_{k}^{x,y} \) must be equal or be before \( r(m) \).

This contradicts the notion that \( e_{k}^{x,y} \) is after \( r(m) \).

Next assume that \( p_{k} \) is the forbidden process \( p_{1} \). Since
\[
e_{k}^{x,y}(= CGCy(x,y,1)) \neq e_{k}^{x,y}, xgcn_{1}(i) = x \at \( e_{k}^{x,y} \).
\]
Thus \( x' = x \) and \( (xgcn_{1}(i), ygcn_{1}(i)) = (x,y') \) \( (y' > 0) \) at \( r(m) \). Therefore, \( p_{1} \) must take additional checkpoint \( e_{k}^{x+1,y} \) before \( r(m) \). This contradicts the notion that \( e_{k}^{x+1,y} \) is after \( r(m) \).

(Case 2: \( e_{k}^{x,y} \) is after \( s(m) \)) Since \( e_{k}^{x,y} = e_{k}^{x+1,y} \), \( p_{1} \) is not the forbidden process.

(Case 2-1: \( e_{k}^{x,y} \rightarrow e_{k}^{x',y} \)) Since \( e_{k}^{x,y} \) is \( p_{1} \)'s newest checkpoint at \( e_{k}^{x,y} \) and \( e_{k}^{x'} \rightarrow e_{k}^{x,y} \rightarrow e_{k}^{x,y} \rightarrow e_{k}^{x,y} \), \( sce_{k}(i) = true \) at \( e_{k}^{x,y} \). Thus, \( e_{k}^{x,y} \) must be the newly taken checkpoint just before \( e_{k}^{x,y} \) from Rule 1. This contradicts the fact that there is an event \( s(m) \) between \( e_{k}^{x,y} \) and \( e_{k}^{x,y} \).

(Case 2-2: \( e_{k}^{x,y} \neq e_{k}^{x,y} \)) Since there is event \( s(m) \) between \( e_{k}^{x,y} \) and \( e_{k}^{x,y} \), \( e_{k}^{x,y} \) is a receive event from a process \( p_{2} \). Let \( (x', y') = (xgcn_{1}(j), ygcn_{1}(j)) \) at \( e_{k}^{x,y} \). \( (x', y') \geq (x, y) \) is satisfied.

First assume that \( p_{k} \) is a normal process. Since \( e_{k}^{x,y} \neq e_{k}^{x,y} \), \( (xgcn_{1}(k), ygcn_{1}(k)) \). \( xgcn_{1}(k), ygcn_{1}(k) \). \( xgcn_{1}(k), ygcn_{1}(k) \) is after \( e_{k}^{x,y} \). Thus, \( (xgcn_{1}(k), ygcn_{1}(k)) \geq (x', y') \) and \( s_{2}(k) \) is true at \( e_{k}^{x,y} \). From Rule 2, \( e_{k}^{x,y} \) must be the newly taken checkpoint just before \( e_{k}^{x,y} \). This contradicts the notion that there is an event \( s(m) \) between \( e_{k}^{x,y} \) and \( e_{k}^{x,y} \).

Next assume that \( p_{k} \) is the forbidden process \( p_{1} \). Since \( xgcn_{1}(1), ygcn_{1}(1) \) is after \( e_{k}^{x,y} \). Since \( y > 0 \), \( (x', y') > (x, y) \) is satisfied. In addition, \( s_{2}(1) \) is true at \( e_{k}^{x,y} \). From Rule 2, \( e_{k}^{x,y} \) must be the newly taken checkpoint just before \( e_{k}^{x,y} \). This contradicts the notion that there is an event \( s(m) \) between \( e_{k}^{x,y} \) and \( e_{k}^{x,y} \). Therefore, \( CGCy(x,y) \) is consistent.

\[ \square \]

4 Multiple forbidden processes

When there are multiple forbidden processes, it is impossible for the forbidden processes to take an additional checkpoint only if the condition of Theorem 1 is satisfied.

Consider the execution in Fig. 4, where \( p_{1} \) and \( p_{2} \) are forbidden processes. \( p_{2} \) does not need to take an additional checkpoint at \( r(m_{2}) \) since the condition of Theorem 1 is not satisfied at \( r(m_{2}) \). \( p_{1} \) does not need to take an additional checkpoint at \( r(m) \) by the same reason. Thus, \( gc(c,1) = \bot \). Consider the following execution after current event. \( p_{3} \) sends message \( m' \) to \( p_{2} \) and \( p_{2} \) executes \( r(m') \). The condition of Theorem 1 is satisfied at \( r(m') \) and \( p_{2} \) takes an additional checkpoint \( c_{2}^{1} \) just before \( r(m') \). \( p_{2} \) sets \( gc(c,2) = c_{2}^{1} \). However, \( \bot \rightarrow c_{2}^{1} \) and \( gc(c) \) is not consistent.

In this execution, \( p_{1} \) needs to take an additional checkpoint just before \( r(m) \). The minimum condition for forbidden processes in the system with multiple forbidden processes to take an additional checkpoint is an open question. In order to attain the minimum condition, the problem of taking the minimum number of additional checkpoints when there are no forbidden processes must be solved, because a system without forbidden processes is the same as one in which all processes are forbidden processes.

5 Concluding remarks

This paper discussed a coordinated checkpointing algorithm for distributed systems with a forbidden process. It described a checkpointing algorithm which minimizes the number of additional checkpoints in the forbidden process. However, the number of additional checkpoints in normal processes is not minimized. Minimizing it remains unsolved. Another unsolved problem is minimizing the number of additional checkpoints in the forbidden processes when there are multiple forbidden processes.

Acknowledgment

I would like to thank Dr. Hirofumi Katsuno of NTT Basic Research Laboratories for his encouragement and suggestions.

References


program MCGC; /* program for $p_i$ */
const $n = ...$; /* number of processes */
/* $p_i$ : forbidden process, $p_2, \ldots, p_n$: normal processes */
var $xgcn(n), ygcn(n), ck(n), ca(*, *)$: integer;
see($n$), st($n$): boolean;

procedure checkpoint
begin
take a checkpoint;
$ck(i) := ck(i) + 1$;
for each $k \neq i$ do $see(k) := true$;
$see(i) := false$;
for each $k$ do $st(k) := false$;
end; /* end of subroutine */

/* main */
initialization begin
for each $k \neq i$ do $ck(k) := -1$;
$ck(i) := 0$;
for each $k \neq 1$ do $xgcn(k) := -1$;
if $i = 1$ then $xgcn(1) := 0$ else $xgcn(1) := -1$;
for each $k \neq 1$ do $ygcn(k) := 1$;
ygcn(1) := 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;
if $i = 1$ /* forbidden process */
then $xgcn(i) := xgcn(i) + 1$
else $ygcn(i) := ygcn(i) + 1$;
$ca(xgcn(i), ygcn(i)) := ck(i)$;
end; /* end of checkpoint initiation */

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

when $p_i$ receives ($m$, $mxgcn$, $mygcn$, $mck$, $msee$) from $p_j$
begin
for each $k$
do
if $ck(k) = mck(k)$ then $see(k) := see(k) \lor msee(k)$
else if $ck(k) < mck(k)$ then $see(k) := msee(k)$;
for each $k$
do
$xgcn(k), ygcn(k) := max((mxgcn(k), mygcn(k)), (xgcn(k), ygcn(k)))$;
for each $k$ do $ck(k) := max(ck(k), mck(k))$;
if ($mxgcn(j), mygcn(j)) > (xgcn(i), ygcn(i))$ then
begin /* information about new initiation */
if $i = 1$ /* forbidden process */
then begin
checkpoint;
xgcn(i) := xgcn(i) + 1;
end else begin /* normal process */
if ($see(i) = true$ or
($3h, st(h) = true$ and
$(mxgcn(j), mygcn(j)) > (xgcn(h), ygcn(h))$) then
checkpoint;
xgcn(i), ygcn(i) := (mxgcn(j), mygcn(j)));
end; /* end of normal process case */
ca(xgcn(i), ygcn(i)) := ck(i);
end; /* end of case new information arrives */
execute $r(m)$;
end; /* end of message receiving */

Figure 5: Algorithm MCGC.