Garbage Collection in Uncoordinated Checkpointing Algorithms

LIU Yunlong (刘云龙) and CHEN Junliang (陈俊亮)

National Laboratory of Switching Technology and Telecommunication Networks
Beijing University of Posts and Telecommunications, Beijing 100876, P.R. China

E-mail: ylliu@technologist.com; chenjl@bupt.edu.cn

Received October 31, 1997; revised January 7, 1998.

Abstract In this paper, the hard problem of the thorough garbage collection in uncoordinated checkpointing algorithms is studied. After introduction of the traditional garbage collecting scheme, with which only obsolete checkpoints can be discarded, it is shown that this kind of traditional method may fail to discard any checkpoint in some special cases, and it is necessary and urgent to find a thorough garbage collecting method, with which all the checkpoints useless for any future rollback-recovery including the obsolete ones can be discarded. Then, the Thorough Garbage Collection Theorem is proposed and proved, which ensures the feasibility of the thorough garbage collection, and gives the method to calculate the set of the useful checkpoints as well.

Keywords distributed system, checkpointing and rollback recovery, storage management, thorough garbage collection

1 Introduction

Checkpointing and rollback recovery (CRR) has been considered as the most effective and flexible approach to fault-tolerance of long-lived distributd systems. It periodically records the states of the processes in the stable storage, an action called checkpointing, and the saved states are called checkpoints. Upon detection of a fault, the system can resume its execution from the latest consistent global checkpoint rather than its beginning, an action called rollback-recovery. Since 1980, CRR schemes in distributed systems have been widely studied by many researchers[1−16]. They can be normally categorized into two classes: coordinated checkpointing and uncoordinated checkpointing. With coordinated checkpointing[1−5], all the processes in a system coordinate their checkpoints to form a system-wide consistent state, in which all the established checkpoints are mutually consistent, and hence the rollback-recovery is very easy to implement. With uncoordinated checkpointing[6−11], each process in the system takes its checkpoints independently, and the system-wide consistent state is computed during the failure-recovery, and hence the rollback-recovery here is much more complex than that with coordinated checkpointing.

Besides the fault-tolerant issues, the storage management is also an important problem in long-lived systems. Specially in the CRR schemes, some storage managing measure must be taken to conrol or limit the total storage overhead of the corresponding protocols because all CRR schemes are based on the redundancy of storage resource (i.e. checkpoints), and the event of the storage overflow would have a strong influence on the fault-tolerance. The studies on garbage collection are just to find the methods of storage management. The purpose of garbage collection is to remove the garbage chekpoints that can never be used by any possible rollback-recovery in the future in order to reclaim the space occupied meaninglessly and prevent the storage from overflowing. If a garbage collection algorithm

Supported by the Doctorate Research Foundation of the State Education Commission of China.

can remove all garbage checkpoints, and any checkpoint left must be useful for some future rollback-recovery, then this garbage collection algorithm is said to be thorough (or clean).

In coordinated algorithms, each process needs only to maintain the latest checkpoint to recover the possible failures, and all earlier checkpoints can be removed easily when a new checkpoint has been taken. However, garbage collection in uncoordinated checkpointing algorithms is not so simple as that in coordinated algorithms. The inherent uncertainty of distributed systems makes it usually impossible to predict the future patterns of message interaction and checkpoint insertion, so it is rather difficult to predict the future rollback-recoveries and their possible affecting scope. Thus, the thorough garbage collection is a very hard problem. The main contribution of this paper is that a formal method to calculate the useful checkpoints for future recoveries is presented, which makes thorough garbage collection feasible.

This paper is organized as follows. Section 2 describes the system model and some necessary terminology. Section 3 introduces the traditional garbage collection scheme which can remove only the obsolete checkpoints. Section 4 proposes and proves the Thorough Garbage Collection Theorem, and presents a flexible and timely garbage collecting algorithm, and Section 5 concludes this paper.

2 Preliminary

We consider the asynchronous message-passing systems in this paper, which can be viewed as an ensemble of N (throughout the paper, we use N to denote the total number of processes in the system) autonomous processes inter-connected by a communication network. There exists no shared memory or common clock among the processes. Processes communicate with each other only by sending message via the corresponding channel.

2.1 Precedence and Precedence-Graph

Let CP_i^x $(0 \leq i < N,\ 0 \leq x)$ denote the x-th checkpoint of the i-th process, where i is called the process id and x the checkpoint index (we assume each process p_i starts its execution with an initial checkpoint CP_i^0), and let CI_i^x $(0 \leq i < N,\ 0 \leq x)$ denote the checkpointing interval (or interval) between CP_i^x and CP_i^{x+1}, where CP_i^x is called the beginning checkpoint of this interval, and CP_i^{x+1} is called the ending checkpoint of the interval. During normal execution, the processes need to interact with each other to transfer computation information. This kind of message exchange will cause dependencies between processes' computation. Similar to Lamport's happened-before in [12], we define:

Definition 1. *Given a checkpoint and communication pattern, for any two checkpointing intervals CI_i^x and CI_j^y, we say that CI_i^x precedes CI_j^y (denoted by $CI_i^x \prec CI_j^y$) if and only if:*

1) $(i = j) \wedge (x < y)$; *or*

2) $i \neq j$, *and there is a message sent by p_i within CI_i^x and received by p_j within CI_j^y*; *or*

3) $\exists CI_k^u.\ (CI_i^x \prec CI_k^u) \wedge (CI_k^u \prec CI_j^y)$.

Moreover, the first two cases are called direct precedence, and the last case is called transmitted precedence. Easy to observe that the relationship of precedence indicates the time order between two intervals, but it is not partially ordered, since it may be symmetrical.

From the precedence relationships created during the execution of a distributed system, we can draw its precedence-graph (or P-graph), denoted by $PG ::= \langle V, E \rangle$, where V denotes the set of the nodes, and E denotes the set of the directed edges. In a P-graph, each node represents a checkpointing interval and each directed edge represents the direct precedence between the connected nodes (the source node of an edge directly precedes the destination

node of it). What worth noting is that, there may exist circles in a P-graph[1], and the executions of different computations may create the same P-graph.

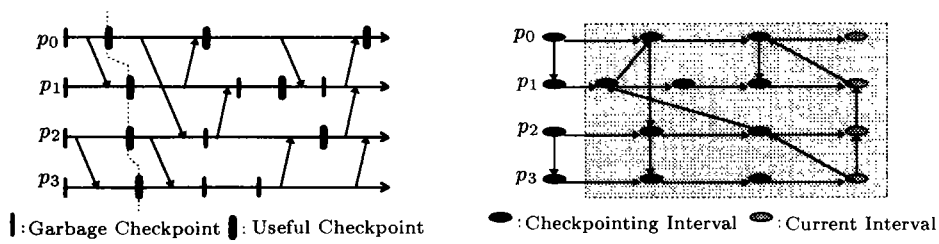
Definition 2. *Given a P-graph $PG ::= \langle V, E\rangle$ and a node subset $S \subseteq V$, all the reachable nodes (including the nodes in S) from any node in S and the passed edges can form a sub-graph of PG, which is called the induced sub-graph from S of the mother graph GP, denoted by $PG(S)$. Formally, the node set of $PG(S)$, denoted by V_S, can be computed as follows:*

$$V_S = \{CI_k^z | (CI_k^z \in S) \vee (\exists CI_j^y \in S : (CI_k^z \in V) \wedge (CI_j^y \prec CI_k^z))\}.$$

Definition 3. *In a given induced sub-graph $PG(S)$, the beginning checkpoints of those checkpointing intervals with the lowest index among all intervals with the same process id compose the local checkpoint set defined by $PG(S)$, denoted by $\lfloor PG(S) \rfloor$. Formally,*

$$\lfloor PG(S) \rfloor = \{CP_i^x | (\forall p_i \in P) \wedge (\exists CI_i^x \in V_S) : \forall CI_i^y \in V_S.(x \leq y)\}.$$

For example, in the checkpoint and communication pattern shown in Fig.1(a), whose precedence graph is shown in Fig.1(b), let $S1$ denote the set of the current checkpointing intervals of all processes, i.e. $S1 = \{CI_0^3, CI_1^4, CI_2^3, CI_3^3\}$, then we can get the induced sub-graph $PG(S1)$, shown as the shadowed part in Fig.1(b). From Definition 3, we have $\lfloor PG(S) \rfloor = \{CP_0^1, CP_1^1, CP_2^1, CP_3^1\}$, as shown by the dashed line in Fig.1(a).



(a) Time-space diagram. (b) P-graph.

Fig.1

2.2 Consistent Global Checkpoint

Definition 4. *A global checkpoint of a distributed system refers to a collection of N local checkpoints, one from each process in the system. Given a global checkpoint, if any two local checkpoints, say CP_i^x and CP_j^y (permitting $CP_i^x = CP_j^y$), in it satisfy the condition: $\neg\exists r \geq x$, $\neg\exists t < y$: $CI_i^r \prec CI_j^t$, then the global checkpoint is said to be consistent. When a failure happens, the most recent one of all the consistent global checkpoints is referred to as the recovery line of this failure.*

Theorem 1 (Recovery Line Theorem). *Given a distributed system consisting of N processes, if there are m $(0 < m \leq N)$ processes that failed concurrently during its execution, then the recovery line of these concurrent failures must be able to be constructed from $\lfloor PG(F) \rfloor$, where F denotes the set of the current checkpointing intervals of the failed processes.*

Proof. We first prove that at least one consistent global checkpoint can be constructed from $\lfloor PG(F) \rfloor$, and then prove that this consistent global checkpoint is the most recent one.

(1) Suppose that no consistent global checkpoint can be constructed from $\lfloor PG(F) \rfloor$. Then, from Definition 4, there must be at least two local checkpoints, say CP_i^x and CP_j^y,

[1]This kind of circle indicates that there exists domino effect in the corresponding execution. For example, in the checkpoint and communication pattern shown in Fig.2, the failure of p_1 will cause the whole system rollback to its initial state. We had once done a deep analysis on domino effect, and given three resolving strategies, that is, domino-avoidance, domino-elimination and domino-tolerance. The interested readers are referred to [16].

in $\lfloor PG(F) \rfloor$ so that $\exists r \geq x$, $\exists t < y$: $CI_i^r \prec CI_j^t$. We have: $r \geq x \overset{Definition\ 1}{\Rightarrow} CI_i^x \prec CI_i^r \prec CI_j^t \overset{Definition\ 2}{\Rightarrow} CI_j^t \in PG(E) \overset{Definition\ 3}{\Rightarrow} t \geq y$, which contradicts the supposition. So, there is at least one consistent global checkpoint (say $CGCP$) that can be constructed from $\lfloor PG(F) \rfloor$.

(2) For any global checkpoint (say GCP') more recent than $CGCP$, we have:

$$\left.\begin{array}{l} \exists p_i \in P : (CP_i^x \in \lfloor PG(F) \rfloor) \wedge (\exists r > x : CP_i^r \in GCP') \\ \exists CI_j^y \in F : CI_j^y \prec CI_i^x \Rightarrow \exists CI_j^t : (t \leq y) \wedge (CI_j^t \in GCP') \end{array}\right\}$$
$$\Rightarrow \exists \langle CP_i^r, CI_j^t \rangle : (\exists x < r, \exists y \geq t) \wedge (CI_j^y \prec CI_i^x),$$

which indicates that all the global checkpoints more recent than $CGCP$ are inconsistent. Thus, $CGCP$ is the most recent one.

Summing up the results of the steps above, we can draw that the theorem is true. □

In the checkpoint and communication pattern shown in Fig.1(a), if p_3 fails in its checkpointing interval CI_3^3, then we can calculate the recovery line of this failure from $PG(\{CI_3^3\})$: $\lfloor PG(\{CI_3^3\}) \rfloor = \{CP_0^1, CP_1^1, CP_2^1, CP_3^1\}$, as shown by the dashed line in the figure.

Definition 5. *Given any two global checkpoints GCP_l and GCP_h, if and only if:*

$$\forall i : 0 \leq i \leq N-1.(index(GCP_l[p_i] \leq index(GCP_h[p_i]),$$

we say that GCP_l is a history of GCP_h, denoted by $GCP_l \trianglelefteq GCP_h$.

The following lemma on the lattice property of the set of all consistent global checkpoints has been proved elsewhere[7].

Lemma 1. *Given a checkpoint and communication pattern, we refer to the set of all consistent global checkpoints as $\{CGCP\}$. Then, $\langle \{CGCP\}, \trianglelefteq \rangle$ forms a lattice. For any GCP_l, $GCP_h \in \{CGCP\}$, the least upper bound and the greatest lower bound of them can be computed as follows:*

(1) $\forall i : 0 \leq i \leq n-1.(index\,(LUB(GCP_l, GCP_h)[p_i]) = \max(index(GCP_l[p_i], index\,(GCP_h[p_i])))$;

(2) $\forall i : 0 \leq i \leq n-1.(index\,(GLB(GCP_l, GCP_h)[p_i]) = \min(index(GCP_l[p_i], index\,(GCP_h[p_i])))$.

Theorem 2. *For a given set of checkpointing intervals $F = \{CI_0, CI_1, \ldots, CI_m\}$, we must have*

$$\lfloor PG(F) \rfloor = GLB(\lfloor PG(CI_0) \rfloor, \lfloor PG(CI_1) \rfloor, \ldots, \lfloor PG(CI_m) \rfloor).$$

Proof.

$$F = \{CI_0, CI_1, \ldots, CI_m\}$$
$$\overset{Definition\ 3}{\Rightarrow} PG(F) = \bigcup_{0 \leq i \leq m} PG(CI_i)$$
$$\overset{Lemma\ 1}{\Rightarrow} \lfloor PG(F) \rfloor = GLB(\lfloor PG(CI_0) \rfloor, \lfloor PG(CI_1) \rfloor, \ldots, \lfloor PG(CI_m) \rfloor) \quad \square$$

Definition 6. *In a given checkpoint and communication pattern, a local checkpoint CP_i^x $(0 \leq i < N,\ 0 < x)$ is said to be invalid if and only if $\exists r < x$, $\exists t \geq x$: $(CI_i^t \prec CI_i^r)$.*

Theorem 3. *The invalid checkpoints cannot be included in any consistent global checkpoint.*

Proof. It is evident to draw this result from Definition 4 and Definition 6. □

3 The Obsolete Checkpoint Based Scheme

Over the past decade, a simple garbage collecting scheme based on the notion of obsolete checkpoint has been used[13–15]. In this scheme, the most recent consistent global checkpoint $\lfloor PG(S) \rfloor$ is computed, where S is the set of the current checkpointing intervals of all processes in the system, and all checkpoints taken before $\lfloor PG(S) \rfloor$ are obsolete and can

be removed; all non-obsolete checkpoints taken after $\lfloor PG(S) \rfloor$ (including the checkpoints in $\lfloor PG(S) \rfloor$) will be retained because they may be used by the possible rollback-recoveries in the future. In the checkpoint and communication pattern shown in Fig.1(a), the most recent consistent global checkpoint is $\lfloor PG(S) \rfloor = \{CP_0^1, CP_1^1, CP_2^1, CP_3^1\}$, so the local checkpoints taken before $\lfloor PG(S) \rfloor$, i.e. $CP_0^0, CP_1^0, CP_2^0, CP_3^0$, are obsolete and will be removed by this garbage collecting scheme.

This kind of traditional garbage collection has the clear advantage of easiness to implement, but in some special cases, it cannot remove even one checkpoint, and hence it is unable to effectively prevent storage from overflowing. For example, if a system executes throughout in the checkpoint and communication pattern shown in Fig.2, then, no matter how many checkpoints have been taken, the most recent consistent global checkpoint at any moment will be the initial state $\{CP_0^0, CP_1^0, CP_2^0\}$ for ever. Thus, none checkpoint will become obsolete, and the obsolete checkpoint based scheme cannot remove even one checkpoint. As a result, the storage overhead will increase unlimitedly, which will finally cause the storage overflow.

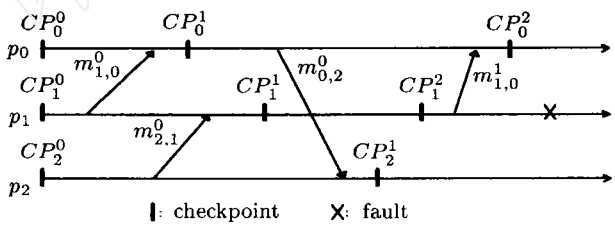


Fig.2

4 The Thorough Scheme

We have shown that the traditional scheme is not effective enough to control the storage overhead, so the thorough garbage collecting scheme is crucial for the realistic applications.

To facilitate the discussion, we define some notions first. Given a checkpoint and communication pattern, let:

- $\Psi(t)$ denotes the set of all local checkpoints in the system that have been taken up to the given moment t;
- $\Psi_O(t) \subset \Psi(t)$ denotes the set of all obsolete checkpoints in the system up to the given moment t;
- $\Psi_I(t) \subset \Psi(t)$ denotes the set of all invalid checkpoints in the system up to the given moment t;
- $\Psi_U(t) \subseteq \Psi(t)$ denotes the set of all local checkpoints in the system useful for some failure-recovery at the given momen t;
- $\Phi(t)$ denotes the set of current checkpointing intervals of all processes in the system at the given moment t;
- $\Psi_N(t) \subseteq \Psi(t)$ denotes the set of all local checkpoints in the system that begin those non-current intervals within which there is no message-receiving event at the given moment t.

Theorem 4 (Thorough Garbage Collection Theorem). *For a checkpoint and communication pattern, at any given moment t, the following are true:*

$T4.1: \Psi_U(t) = \cup_{\forall \lambda \in \Phi(t)} \lfloor PG(\{\lambda\}) \rfloor$;

$T4.2: \forall \alpha \in \Psi(t), \forall \tau > 0: \alpha \in \Psi_U(t+\tau) \Rightarrow \alpha \in \Psi_U(t)$;

$T4.3: (\Psi_O(t) \cap \Psi_U(t) = \phi) \wedge (\Psi_I(t) \cap \Psi_U(t) = \phi) \wedge (\Psi_N(t) \cap \Psi_U(t) = \phi)$;

$T4.4: \Psi_U(t) = \Psi(t) - (\Psi_O(t) \cup \Psi_I(t) \cup \Psi_N(t))$.

Proof. T4.1: From the definition of $\Psi_U(t)$, we have $\Psi_U(t) \equiv \cup_{S \in \Phi(t)^*} \lfloor PG(S) \rfloor$, where $S \in \Phi(t)^*$ is the closure of $\Psi(t)$, which includes all the possible cases of the concurrent

failures at the moment t. From Theorem 2, we have

$$\left.\begin{array}{l}\forall S \in \Phi(t)^* : \forall \alpha \in \lfloor PG(S) \rfloor \Rightarrow \exists \lambda \in S : \alpha \in \lfloor PG(\{\lambda\}) \rfloor \\ \forall S \in \Phi(t)^* \Rightarrow S \subseteq \Phi(t)\end{array}\right\}$$

$$\Rightarrow (\forall \alpha : \alpha \in \cup_{S \in \Phi(t)^*} \lfloor PG(S) \rfloor \Rightarrow \alpha \in \cup_{\lambda \in \Phi(t)} \lfloor PG(\{\lambda\}) \rfloor) \quad (1)$$

$$\forall \lambda : \lambda \in \Phi(t).(\{\lambda\} \in \Phi(t)^*)$$

$$\Rightarrow \forall \alpha : \alpha \in \cup_{\lambda \in \Phi(t)} \lfloor PG(\{\lambda\}) \rfloor \Rightarrow \alpha \in \cup_{S \in \Phi(t)^*} \lfloor PG(S) \rfloor \quad (2)$$

Summing up (1) and (2), we can draw that $\Psi_U(t) \equiv \cup_{S \in \Phi(t)^*} \lfloor PG(S) \rfloor = \cup_{\lambda \in \Phi(t)} \lfloor PG(\{\lambda\}) \rfloor$ is true.

T4.2: Given any local checkpoint $\alpha \in \Psi(t)$, suppose α starts the checkpointing interval λ, then $\alpha \in \Psi_U(t+\tau) \Rightarrow \alpha \in \cup_{S \in \Phi(t+\tau)^*} \lfloor PG(S) \rfloor \overset{T4.1}{\Rightarrow} \exists \mu \in \Phi(t+\tau) : \alpha \in \lfloor PG(\{\mu\}) \rfloor \Rightarrow (\mu = \lambda) \vee (\mu \prec \lambda)$, if

(1) $(\mu = \lambda)$, then we have $(\lambda \in \Phi(t)) \wedge (\mu \in \Phi(t))$, and $\alpha \in \Psi_U(t+\tau) \Rightarrow \alpha \in \Psi_U(t)$ is obviously true;

(2) $(\mu \prec \lambda)$, and if $(\lambda \in \Phi(t)) \vee (\mu \in \Phi(t))$, then $\alpha \in \Psi_U(t+\tau) \Rightarrow \alpha \in \Psi_U(t)$ is obviously true. Otherwise, that is $(\lambda \notin \Phi(t)) \wedge (\mu \notin \Phi(t))$, then the precedence $(\mu \prec \lambda)$ must be created by the relationship transmission, so $\exists \eta \in \Phi(t) : (\mu \prec \eta \prec \lambda)$, and from the definition of P-graph and Definition 3, we have $\alpha \in \lfloor PG(\{\mu\}) \rfloor \Rightarrow \alpha \in \lfloor PG(\{\eta\}) \rfloor$, hence $\alpha \in \Phi(t)$.

Summing up the results of (1) and (2), we can draw that $\forall \alpha \in \Psi(t)$, $\forall \tau > 0 : \alpha \in \Psi_U(t+\tau) \Rightarrow \alpha \in \Psi_U(t)$ is true.

T4.3: $(\Psi_O(t) \cap \Psi_U(t) = \phi)$ is obviously true; $(\Psi_I(t) \cap \Psi_U(t) = \phi)$ is also true following Theorem 3; below, we prove that $(\Psi_N(t) \cap \Psi_U(t) = \phi)$ is true.

Suppose $(\Psi_N(t) \cap \Psi_U(t) = \phi)$ is false, then $\exists \alpha : (\alpha \in \Psi_N(t)) \wedge (\alpha \in \Psi_U(t))$. Suppose α begins the interval μ and ends the intefval η, then we have

$$\alpha \in \Psi_U(t) \Rightarrow \exists \lambda \in \Phi(t) : \alpha \in \lfloor PG(\{\lambda\}) \rfloor \Rightarrow \lambda \prec \mu \quad (1)$$

From the definition of $\Psi_O(t)$, we know that there is no message-receiving event during μ, then we must have

$$(\lambda \prec \eta \prec \mu) \Rightarrow \alpha \notin \lfloor PG(\{\lambda\}) \rfloor \quad (2)$$

The results of the two steps above conflict with each other, so the supposition is false, and hence $(\Psi_N(t) \cap \Psi_U(t) = \phi)$ is true.

Summing up, we can draw that T4.3 is true.

T4.4: Suppose that $\Psi_U(t) = \Psi(t) - (\Psi_O(t) \cup \Psi_I(t) \cup \Psi_N(t))$ is false, then we have

$$\exists \alpha : (\alpha \in \Psi(t)) \wedge (\alpha \notin \Psi_U(t)) \wedge (\alpha \notin (\Psi_O(t) \cup \Psi_I(t) \cup \Psi_N(t)),$$

and

$$\alpha \notin \Psi_U(t) \Rightarrow \forall \lambda \in \Phi(t) : \alpha \notin \lfloor PG(\{\lambda\}) \rfloor.$$

For $\alpha \notin \Psi_O(t)$, α must be after $\lfloor PG(\{\lambda\}) \rfloor$. Suppose α begins the interval μ, then

$$\forall \lambda \in \Phi(t) : \begin{cases} (\lambda \prec \mu \overset{\alpha \notin \Psi_U(t)}{\Rightarrow} \exists \eta : \lambda \prec \eta \prec \mu) & (1) \\ (\lambda = \mu \overset{\alpha \notin \Psi_U(t)}{\Rightarrow} \exists \eta : \lambda \prec \eta \prec \mu) & (2) \end{cases}$$

where η is some interval after $\lfloor PG(\{\lambda\}) \rfloor$ but before μ in the same process as μ. From Definition 1, we can draw that case (1) in which the precedences of μ can be created only through some interval with a lower index in the same process indicates that there is no message-receiving event in μ, and hence $\alpha \in \Psi_N(t)$; in case (2), we have $\mu \prec \eta \prec \mu$, and hence $\alpha \in \Psi_I(t)$. Both two cases contradict the supposition, so the supposition is false. Thus, $\Psi_U(t) = \Psi(t) - (\Psi_O(t) \cup \Psi_I(t) \cup \Psi_N(t))$ is true. □

In Theorem 4, T4.2 indicates that all local checkpoints useful for any future rollback-recovery are predictable, which proves the feasibility of the thorough garbage collection. Furthermore, T4.1 gives the calculating method of the useful checkpoints at any given moment. So, the hard problem of thorough garbage collection is solved herein. Besides, T4.3 and T4.4 together bring to light the composing subsets of the set of garbage checkpoints and their implications.

Based on the discussion above, at any given moment t, only the useful checkpoints included in $\Psi_U(t) = \cup_{\forall\lambda\in\Phi(t)}\lfloor PG(\{\lambda\})\rfloor$ should be retained, and all other checkpoints that are included in $\Psi_O(t) \cup \Psi_I(t) \cup \Psi_N(t)$ should be discarded safely. Besides, during a failure-recovery, all checkpoints after the recovery line should also be discarded because they are no longer meaningful for the resumed computation. For example, in the checkpoint and communication pattern shown in Fig.1(a), we are able to calculate the set of useful checkpoints using T4.1 in Theorem 4, i.e., $\Psi_U(t) = \cup_{\forall\lambda\in\Phi(t)}\lfloor PG(\{\lambda\})\rfloor = \{CP_0^1, CP_0^2, CP_0^3, CP_1^1, CP_1^3, CP_2^1, CP_2^3, CP_3^1\}$, all checkpoints in which are surely useful for some future recovery. Those in $\Psi(t) - \Psi_U(t) = \{CP_0^0, CP_1^0, CP_1^2, CP_1^4, CP_2^0, CP_2^2, CP_3^0, CP_3^2, CP_3^3\}$ are nine garbage checkpoints that ought to be discarded. In the same checkpoint and communication pattern, if the traditional garbage collecting scheme is used, only the four obsolete checkpoints in $\Psi_O(t) = \{CP_0^0, CP_1^0, CP_2^0, CP_3^0\}$ can be removed. Furthermore, for the checkpoint and communication pattern in Fig.2, we have shown that with the traditional scheme, none of the existing checkpoints can be removed, but with the thorough garbage collection, only those in $\Psi_U(t) = \cup_{\forall\lambda\in\Phi(t)}\lfloor PG(\{\lambda\})\rfloor = \{CP_0^0, CP_1^0, CP_2^0\}$ will be retained, and all other checkpoints, i.e. those in $\Psi(t) - \Psi_U(t) = \{CP_0^1, CP_0^2, CP_1^1, CP_1^2, CP_2^1\}$ will be removed without hesitation. Easy to observe that the thorough garbage collection is much more effective than the traditional scheme which is based on the notion of obsolete checkpoints.

Theorem 5. *In a distributed system consisting of N processes, at any given time t, there are at most N local checkpoints useful for future rollback-recoveries in each process.*

Proof. From T4.1 of Theorem 4, we can draw that, in the worst case, that is, in the case that the failure of each current checkpointing interval in $\Phi(t)$ needs different local checkpoints of each process, the set of useful checkpoints $\Psi_U(t) = \bigcup\limits_{\forall\lambda\in\Phi(t)}\lfloor PG(\{\lambda\})\rfloor$ includes at most N local checkpoints of each process. □

Theorem 5 tells the lowest upper bound of the storage overhead of the uncoordinated checkpointing algorithms with the thorough garbage collection. This bound can be used to compose the threshold condition of starting the garbage collecting operation. In details, we can define a threshold condition function: $K = f(n)$ so that the thorough garbage collection task can be called whenever the number of local checkpoints in some process exceeds this threshold K. This threshold K can be adjusted under the function to satisfy the special storage requirements of the concrete applications. Thus, a simple, effective, dynamic and timely management of storage in the uncoordinated checkpointing algorithms can be implemented.

5 Conclusion

A garbage collection is said to be thorough or clean if it can remove all the useless checkpoints and retain all the useful ones for future rollback-recoveries. In this paper, we have solved the hard problem of the thorough garbage collection in uncoordinated checkpointing. First, some necessary terms are defined, such as the precedence relationship between checkpointing intervals, the precedence-graph and its induced sub-graph, based on which consistent global checkpoint can be further defined and calculated. Next, the obsolete checkpoints based garbage collection is shown to be not effective enough to prevent storage from overflowing, and there is a high necessity to find a thorough garbage collecting method which

can effectively limit the storage overhead of uncoordinated checkpointing to the minimum. Based on this necessity, the Thorough Garbage Collection Theorem is proposed and proved. This theorem gives a systematic and sound insight to the hard problem mentioned above. Finally, a flexible dynamic threshold condition to start the thorough garbage collection is discussed too.

References

[1] Chandy K M, Lamport L. Distributed snapshot: Determining global states of distributed systems. *ACM Trans. Computer Systems,* May 1985, 3: 63–75.

[2] Koo R, Toueg S. Checkpoint and rollback-recovery for distributed systems. *IEEE Trans. Software Engineering,* Jan. 1987, 13(1): 23–31.

[3] Cristian F, Jahanian F. A timestamp-based checkpointing protocol for long-lived distributed computations. In *Proc. 10th Symp. Reliable Distributed Systems,* 1991, pp.12–20.

[4] Tamir Y, Sequin C H. Error recovery in multiprocessors using global checkpoints. In *Proc. Int. Conf. Parallel Processing,* 1984, pp.32–41.

[5] Ramanathan P, Shin K G. Use of common time base for checkpointing and rollback recovery in a distributed system. *IEEE Trans. Software Engineering,* June 1993, 19(6): 571–583.

[6] Bhargava B, Lian S R. Independent checkpointing and concurrent rollback for recovery in distributed systems — An optimistic approach. In *Proc. 7th IEEE Symp. Reliable Distributed Syst.,* Oct. 1988, pp.3–12.

[7] Wang Y M *et al.* Consistent global checkpoints based on direct dependency trackability. *Information Processing Letters,* May 1994, 50(4): 223–230.

[8] Baldon R *et al.* Characterization of consistent global checkpoints in large-scale distributed systems. In *Proc. the 5th IEEE Workshop on Future Trends of Distributed Computing Systems,* Aug. 1995, Korea, pp.314–323.

[9] Kim Junguk L, Park Taesoon. An effeicient protocol for checkpointing recovery in distributed systems, *IEEE Trans. Parallel and Distributed Systems,* Aug. 1993, 4(8): 955–960.

[10] Prakash Ravi, Singhal Mukesh. Low-cost checkpointing and failure recovery in mobile computing systems. *IEEE Trans. Parallel and Distributed Systems,* Oct. 1996, 7(10): 1035–1048.

[11] Sean W Smith *et al.* Completely asynchronous optimistic recovery with minimal rollbacks. In *Proc. IEEE FTCS-25,* 1995, pp.361–370.

[12] Lamport L. Time, clocks, and the ordering of events in a distributed system. *Comm. ACM,* 1978, 21(7): 558–565.

[13] Strom R E, Yemini S. Optimistic rocovery in distributed systems. *ACM Trans. Computer System,* Aug. 1985, 3(3): 204–226.

[14] Tsuruoka K *et al.* Dynamic recovery schemes for distributed systems. In *Proc. IEEE 2nd Symp. Reliability in Distr. Software and DataBase Syst.,* 1981, pp.124–130.

[15] Chiu Ge-ming, Young Cheng-ru. Efficient rollback-rocovery technique in distributed computing systems, *IEEE Trans. Parallel and Distributed Systems,* June 1996, 7(6): 565–577.

[16] Liu Yunlong, Chen Junliang. Study on resolutions of Domino effects. *Chinese Journal of Software,* Dec. 1998, 12: 942-945.

LIU Yunlong was born in 1972. He received his Bachelor degree in computer science from Harbin Engineering University in 1993 and his Ph.D. degree in communication and information systems from Beijing University of Posts and Telecommunications in 1998. Now, he is working as a Member of Technical Staff in Bell Laboratories — Lucent Technologies. His Research interests include fault tolerant computing, intelligent network, telecommunication management network, and heterogeneous networks interworking. Address: Bell Labs, 3F, Huilong Technical Center, 30 Haidian Nan Lu, Beijing, 100080, P.R. China.