

Adaptive Communication-Induced Checkpointing Protocols with Domino-Effect Freedom

JICHIANG TSAI, CHI-YI LIN* AND SY-YEN KUO*

*Department of Electrical Engineering
National Chung Hsing University
Taichung, 402 Taiwan*

**Department of Electrical Engineering
National Taiwan University
Taipei, 106 Taiwan*

The *domino effect* is an important problem for the checkpointing and rollback recovery in distributed systems. *Communication-induced checkpointing* is one way of preventing domino effect. Most existing such protocols focus on guaranteeing that every checkpoint is part of a consistent global checkpoint. This may induce high run-time overhead due to the possibly excessive number of extra forced checkpoints. In this paper, we propose several adaptive communication-induced checkpointing protocols with domino-effect freedom. These protocols allow a flexible tradeoff between the cost of checkpoint coordination and the rollback distance. Only a specific set of checkpoints needs to be part of a consistent global checkpoint. The overhead analysis shows that our generalization can significantly reduce the number of extra forced checkpoints.

Keywords: distributed systems, domino effect, communication-induced checkpointing, fault tolerance, rollback recovery

1. INTRODUCTION

A distributed computation consists of a finite set of processes that communicate and synchronize with each other by exchanging messages through a network. A local checkpoint is a snapshot of the local state of a process, saved on nonvolatile storage in order to survive process failures. It can be reloaded into volatile memory in case of a failure to reduce the amount of lost work. When a process records such a local state, we say that this process takes a (local) checkpoint. The set of messages and the set of local checkpoints form the *checkpoint and communication pattern* associated with the distributed computation. A global checkpoint M [1] is a set of local checkpoints, one from each process; M is *consistent* if no message is sent after a checkpoint in M and received before another checkpoint in M [2].

If local checkpoints are taken independently, there is a risk that no consistent global checkpoint can be formed from them. This is the problem known as the *domino effect* [3], in which unbounded, cascading rollback propagation can occur during the process of finding a consistent global checkpoint. Many protocols have been proposed to selectively

Received December 27, 2001; revised March 25, 2003; accepted April 10, 2003.
Communicated by Chu-Sing Yang.

take local checkpoints to eliminate possibility of the domino effect (see [4]). Coordinated checkpointing [2, 5] is one way of avoiding the domino effect by synchronizing the checkpointing actions of all processes through explicit control messages. In contrast, *communication-induced checkpointing protocols* [6] achieve coordination by piggy-backing control information on application messages. In addition to taking application-specific *basic* checkpoints, each process can also be asked by the protocol to take additional *forced* checkpoints, based on the piggybacked information as well as local control variables.

Most communication-induced checkpointing protocols proposed in the literature [7-13] are designed to add forced checkpoints to ensure that every checkpoint can be part of a consistent global checkpoint. However, this scenario may induce high run-time overhead due to the possibility of an excessive number of extra checkpoints. In many applications, it is not necessary to guarantee that no cascading rollback will occur. Particularly, in case of a failure a recovery that will not lead to cascading rollback beyond a certain extent is acceptable. Hence we do not need to make every checkpoint be part of a consistent global checkpoint. It is sufficient to consider only a specific set of checkpoints. In [14], Wang and Fuchs first generalized the concept of checkpoint coordination by introducing the notion of *laziness* Z as a measure of the frequency for performing coordination. Only corresponding checkpoints with ordinal numbers nZ , where n is an integer, are required to be consistent with each other for bounding rollback propagation. Overhead analysis showed that their generalization can significantly reduce the number of extra forced checkpoints compared to a previous work [7], which corresponds to the case $Z = 1$.

The contribution of this paper is to introduce a general adaptive communication-induced checkpointing protocol with domino-effect freedom. Such a protocol provides a flexible tradeoff between cost of checkpoint coordination and rollback distance. Similar to the concept of lazy checkpoint coordination [14], only the last checkpoint of a process with timestamp not larger than nK , where n is an integer and K is a measure of the frequency for performing coordination, are required to be part of a consistent global checkpoint. Moreover, the proposed protocol can define a family of adaptive checkpointing protocols with domino-effect freedom. If we eliminate some of its control information, the general protocol gives rise to some particular protocols. The protocol proposed in [14] can also be obtained as a special case. Actually, these protocols differ only in the conditions of directing processes to take forced checkpoints. Last but not least, we show by overhead analysis that the idea of adaptive checkpoint coordination is viable for significantly reducing the number of forced checkpoints compared to the previous works [7-9], that correspond to the case $K = 1$.

This paper is divided into five main sections. Section 2 defines the computational model and describes definitions of *Z-paths* and *Z-cycles*. In section 3, we begin to design the general adaptive checkpointing protocol. In section 4, we show how to implement such a general protocol and also derive a family of checkpointing protocols from it. Section 5 gives some analyses for this family of checkpointing protocols and related work in the literature. Finally, we conclude the paper in section 6.

2. PRELIMINARIES

2.1 Checkpoint and Communication Patterns

A distributed computation consists of a finite set P of N processes $\{P_1, P_2, \dots, P_N\}$ which communicate and synchronize only by exchanging messages. We assume that each pair of processes is connected by a reliable, asynchronous channel with unpredictable but finite transmission delays. Processes fail according to the fail-stop model.

A process can execute *internal*, *send* and *receive* statements. An internal statement does not involve any communication. When P_i executes the statement “*send*(m) to P_j ”, it puts message m into the channel from P_i to P_j . When P_i executes the statement “*receive*(m)”, it is blocked until at least one message directed to P_i has arrived, after which a message is delivered to P_i . Executions of internal, send and receive statements are modeled by internal, sending and receiving events, respectively.

The execution of each process produces a sequence of events, and all the events produced by a distributed computation can be modeled as a partially ordered set with the well-known Lamport’s *happened-before* relation “ \underline{hb} ”, defined as follows [15]:

Definition 1 The relation “ \underline{hb} ” on the set of events satisfies the following conditions:

- (1) If a and b are events of the same process and a comes before b , then $a \underline{hb} b$.
- (2) If a is the event *send*(m) and b is the event *receive*(m), then $a \underline{hb} b$.
- (3) If $a \underline{hb} b$ and $b \underline{hb} c$ then $a \underline{hb} c$.

Given a distributed computation H , its associated checkpoint and communication pattern consists of the set of messages and the set of local checkpoints in H . Fig. 1 shows an example checkpoint and communication pattern. $C_{i,x}$ represents the x th checkpoint of process P_i , where i is called the process id and x , the *index* of this checkpoint. The sequence of events occurring at P_i between $C_{i,x-1}$ and $C_{i,x}$ ($x > 0$) is called a *checkpoint interval* and is denoted by $I_{i,x}$. Each process P_i is assumed to start its execution with an initial checkpoint $C_{i,0}$.

A message m sent by process P_i to process P_j is called *orphan* with respect to the ordered pair of local checkpoints $(C_{i,x}, C_{j,y})$ if and only if the receiving event of m occurs before $C_{j,y}$ but its sending event occurs after $C_{i,x}$. An ordered pair of local checkpoints is *consistent* iff there are no orphan messages with respect to this pair. For example, Fig. 1 shows that the pair $(C_{i,1}, C_{j,1})$ is consistent, while the pair $(C_{j,2}, C_{k,1})$ is inconsistent due to the orphan message m_5 .

A *global checkpoint* is a set of local checkpoints, one from each process. Moreover, a global checkpoint is *consistent* iff all its pairs of local checkpoints are consistent. Fig. 1 shows that $(C_{i,1}, C_{j,2}, C_{k,2})$ is a consistent global checkpoint, and because of the inconsistent pair $(C_{j,2}, C_{k,1})$, the global checkpoint $(C_{i,1}, C_{j,2}, C_{k,1})$ is not consistent.

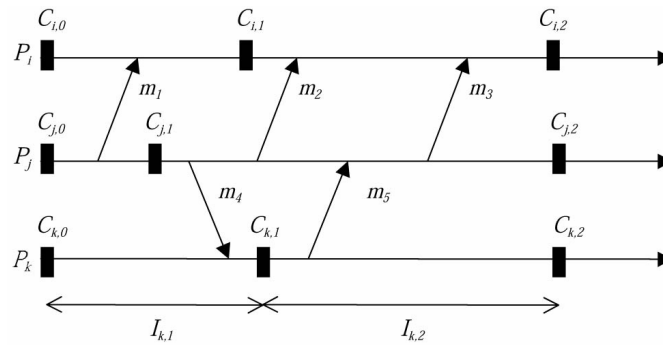


Fig. 1. A checkpoint and communication pattern.

2.2 Z-paths and Z-cycles

Netzer and Xu introduced the following notion of *Z-paths* and *Z-cycles*, and showed that a checkpoint $C_{i,x}$ cannot be part of a consistent global checkpoint iff it is involved in a *Z-cycle* [16].

Definition 2 A *Z-path* is a sequence of messages $[m_1, m_2, \dots, m_q]$ ($q \geq 1$) such that, for each i , $1 \leq i \leq q - 1$: $receive(m_i) \in I_{k,s} \wedge send(m_{i+1}) \in I_{k,t} \wedge s \leq t$.

If a *Z-path* $[m_1, m_2, \dots, m_q]$ satisfies the condition that $C_{i,x}$ precedes $send(m_1)$ and $receive(m_q)$ precedes $C_{j,y}$ in the same process, we say that this *Z-path* is from $C_{i,x}$ to $C_{j,y}$. A *Z-path* from a local checkpoint $C_{i,x}$ to the same local checkpoint $C_{i,x}$ is called a *Z-cycle*. We say that it involves the local checkpoint $C_{i,x}$.

Theorem 1 A local checkpoint $C_{i,x}$ cannot be part of a consistent global checkpoint iff it is involved in a *Z-cycle*.

For example, in Fig. 1, both message sequences $[m_5, m_2]$ and $[m_5, m_3]$ constitute *Z-paths* from $C_{k,1}$ to $C_{i,2}$. Moreover, according to Theorem 1, $C_{k,1}$ cannot be part of a consistent global checkpoint because it is involved in the *Z-cycle* $[m_5, m_4]$.

Since *Z-cycles* are the cause of the domino effect, it is clear that any checkpointing protocol must eliminate all *Z-cycles* by forcing additional checkpoints to ensure domino-effect freedom. We say that all *Z-cycles* are *broken*. For instance, we can add a forced checkpoint before receiving m_5 to break the only *Z-cycle* $[m_5, m_4]$ to make the pattern in Fig. 1 domino-effect free since $[m_5, m_4]$ is no longer a *Z-path*. Now a previously known characterization of consistent global checkpoints corresponding to *Z-paths* is described. This result has been stated and proved in [9] and [16].

Theorem 2 A global checkpoint S is consistent iff there exist no two local checkpoints X and Y in S such that there is a *Z-path* from X to Y .

We say that a *Z-path* is *causal* if the receiving event of each message (except for the last one) precedes the sending event of the next message in the sequence. A *Z-path* is

non-causal if it is not causal. A *Z-path* with only one message is trivially causal. For simplicity, a causal *Z-path* is also called a *causal path*. As an example, the *Z-path* $[m_5, m_3]$ in Fig. 1 is causal, and the *Z-path* $[m_5, m_2]$ is non-causal. For the rest of this paper, we use the following notation: the first (last) message of a *Z-path* ζ is denoted by ζ_{first} (ζ_{last}). Given two *Z-paths* ζ_1 and ζ_2 , if their concatenation is also a *Z-path* then we denote the concatenation as $\zeta_1 \cdot \zeta_2$.

3. DESIGN OF THE GENERAL PROTOCOL

In this section, we begin to design a general adaptive checkpointing protocol with domino-effect freedom, which will make the last checkpoint of a process with timestamp not greater than nK , for some positive integer n , be part of a consistent global checkpoint. We call such a protocol **GPK** (General Protocol with the measure K). Applying Theorem 1, **GPK** must prevent the last checkpoint of a process with timestamp not larger than nK from being involved in a *Z-cycle*. One approach to avoiding generating a *Z-cycle* is to have a checkpoint timestamping algorithm that guarantees the *timestamps for checkpoints always increase along a Z-path* [8, 9]. We can generalize this concept to obtain the following design strategy of the protocol **GPK**.

Design Strategy Let the timestamp of a checkpoint C be denoted as $C.t$. For every pair of checkpoints $C_{i,x}$ and $C_{j,y}$ such that there is a *Z-path* from $C_{i,x}$ to $C_{j,y}$, assure that $C_{j,y}.t > lK$ where $l = \lfloor C_{i,x}.t / K \rfloor$.

In section 5, we demonstrate that exploiting the foregoing strategy accompanied with the basic timestamp described in the following subsection can make the last checkpoint of a process with timestamp not greater than nK be part of a consistent global checkpoint.

3.1 Basic Timestamp Management

Now we derive a new management of timestamps from the classical method [15]. We assume each process P_i has a local logical clock lc_i , which is managed in the following way:

- The variable lc_i is initialized to be 0.
- Before taking a (basic or forced) checkpoint, P_i increments lc_i by 1 and assigns the new value to be the timestamp of the checkpoint.
- Upon sending a message m , P_i assigns $\lfloor lc_i / K \rfloor K$ as the timestamp of m (let $m.t$ denote the timestamp of m).
- Upon receiving a message m , if $m.t > lc_i$, P_i updates lc_i to $m.t$.

Note that for the special case $K = 1$, the previous timestamp management is equivalent to that in [15]. It can be easily verified that this basic timestamp management guarantees that for two local checkpoints $C_{i,x}$ and $C_{j,y}$, if $i = j$ and $x < y$ or if there is a causal

path from $C_{i,x}$ to $C_{j,y}$, the timestamps of the two checkpoints satisfy the design strategy. Below, we examine the case of non-causal paths.

3.2 Z_K -Cycles

Before considering the general management of non-causal paths, we first examine a special case, Z_K -cycles, which are defined as follows:

Definition 3 A Z_K -cycle $[m_1, m_2, \dots, m_q]$ is a Z -cycle composed of two causal paths such that for some positive integer n , message m_1 is sent by process P_i after $lc_i \geq nK$ and message m_q is received by process P_i before a local checkpoint A with $A.t \leq nK$.

Fig. 2 illustrates some examples of Z_K -cycles. We show that Z_K -cycles violate the design strategy in the following theorem.

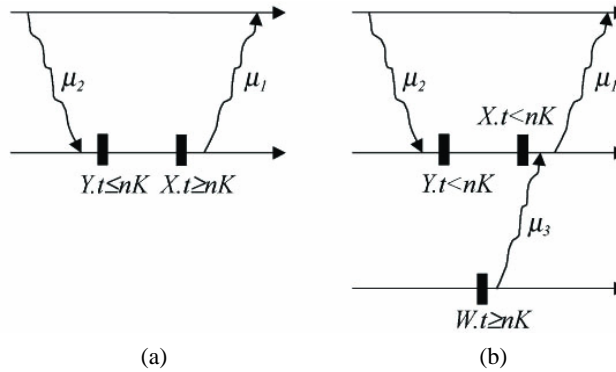


Fig. 2. Some examples of Z_K -cycles.

Theorem 3 A Z_K -cycle Z has to be broken based on the design strategy.

Proof: Suppose Z is the concatenation of two causal paths μ_1 and μ_2 , two cases are considered.

Case a. If Z is from a checkpoint X with $X.t \geq nK$ to a checkpoint Y with $Y.t \leq nK$ for some positive integer n (note that checkpoint X and checkpoint Y may be the same checkpoint), similar to the scenario of Fig. 2 (a), it obviously does not satisfy the design strategy and must be broken;

Case b. Otherwise, Z is from a checkpoint X with $X.t < nK$ to a checkpoint Y with $Y.t < nK$. Checkpoint X and checkpoint Y may be the same checkpoint. According to the definition of Z_K -cycles, message μ_1 .first is sent after $lc_i \geq nK$. Now we represent the next checkpoint of X as $next(X)$. For the sake of $X.t < nK$ and the basic timestamping management, we can find a checkpoint W with $W.t \geq nK$, and there is a causal path μ_3 from W to $next(X)$ with message μ_3 .last received by process P_i before $send(\mu_1$.first), as depicted in Fig. 2 (b). The Z -path $\mu_3 \cdot \mu_1 \cdot \mu_2$ violates the

design strategy. Therefore, a forced checkpoint has to be taken before the event of $receive(\mu_1.last)$, and Z is also broken by this checkpoint.

Hence, a Z_K -cycle Z has to be broken based on the design strategy. □

From the previous theorem, we have that protocol **GPK** has to break every Z_K -cycle by taking a forced checkpoint.

3.3 Management of Non-Causal Paths

Given the basic timestamp management, let us look at every Z -path in Figs. 3 and 4, which contains only two messages m_1 and m_2 with $send(m_2) \text{ hb } receive(m_1)$. $C_{j,y}$ is a local checkpoint taken by P_j before $send(m_1)$ and $C_{k,z}$ is the first checkpoint of P_k taken after $receive(m_2)$. Similar to [9], in order to utilize the information that P_i has about the values of local clocks of other processes, for each k ($1 \leq k \leq N$), we denote by $cl_i[k]$ the value of P_k 's local clock as known by P_i . Remark that P_i can obtain this information with a classical piggybacking technique. If $k = i$, then $cl_i[i] = lc_i$. For $k \neq i$, the knowledge of P_k 's local clock by P_i is only an approximation such that $cl_i[k] \leq lc_k$. To ensure that checkpoint timestamps along non-causal Z -paths satisfy the design strategy, we perform a case analysis to examine whether to force a checkpoint in different scenarios. That is, we have to check if $C_{k,z}.t > m_1.t$. Here, let the value of $m_1.t$ be lK . When P_i receives m_1 , two scenarios can occur. First, let us consider the scenario " $m_1.t > lc_i$ ". In this scenario we have $lc_i < lK$ and thus $m_2.t < lK$. In [9] the value of $cl_i[k]$ is classified as two possible cases, $cl_i[k] < C_{k,z}.t$ and $cl_i[k] \geq C_{k,z}.t$. Now we also discuss these two cases for the value of $cl_i[k]$.

- The value of $cl_i[k]$ has been brought to P_i by a causal path that started from P_k before $C_{k,z}$. This case is illustrated in Fig. 3. More precisely, $cl_i[k]$ is brought to P_i by μ_1 in Fig. 3 (a) and by $\mu_2 \cdot [m_1]$ in Fig. 3 (b). For such a case, we have that $cl_i[k]$ is less than $C_{k,z}.t$, and so two subcases are considered.
- If $cl_i[k] \geq lK$, this means that $C_{k,z}.t > lK$. This subcase obviously satisfies the design strategy.
- If $cl_i[k] < lK$, the timestamp of $C_{k,z}$ may be less than lK since $m_2.t < lK$, too. Hence a safe strategy is to take a forced checkpoint before delivering m_1 so that the sequence $[m_1, m_2]$ is no longer a Z -path.

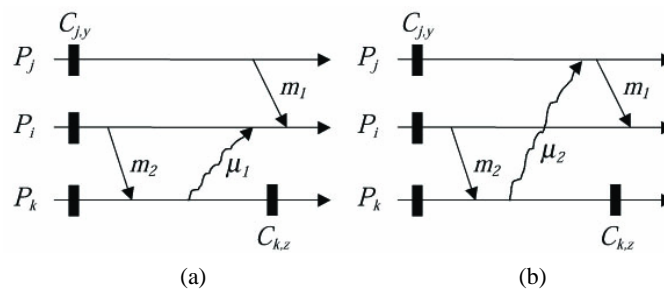


Fig. 3. The value of $cl_i[k] < C_{k,z}.t$.

- The other case is that the value of $cl_i[k]$ has been brought to P_i by a causal path that started from P_k after $C_{k,z}$. This case is depicted in Fig. 4. More precisely, the relevant causal path is μ_1 in Fig. 4 (a) and $\mu_2 \cdot [m_1]$ in Fig. 4 (b). In this case, $cl_i[k]$ is an upper bound of $C_{k,z,t}$. So, if $cl_i[k] < lK$, $C_{k,z,t}$ is less than lK , and a forced checkpoint is necessary for this condition. Hence we only need to consider the situation of $cl_i[k] \geq lK$. There are two subcases to be examined.
- Let us first consider the situation of Fig. 4 (a). If $cl_i[k] \geq lK$, then because the Z -cycle $\mu_1 \cdot [m_2]$ has not been broken, we can conclude that $C_{k,z,t} > lK$. Otherwise, if $C_{k,z,t} \leq lK$, then $\mu_1 \cdot [m_2]$ is a Z_K -cycle, and has to be broken. This leads to a contradiction.
- Now we examine the situation shown in Fig. 4 (b). Again if $cl_i[k] \geq lK$, the Z -cycle $\mu_2 \cdot [m_1, m_2]$ will form a Z_K -cycle when the statement “ $C_{k,z,t} \leq lK$ ” is true. This scenario has been broken by a forced checkpoint when checking the condition of Z_K -cycles. Hence it is unnecessary to bother with it again.

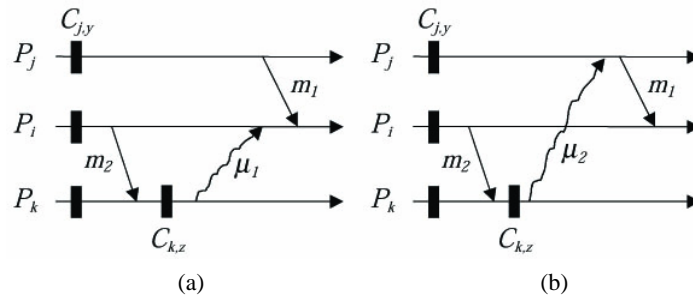


Fig. 4. The value of $cl_i[k] \geq C_{k,z,t}$.

From the previous discussion, we conclude that protocol **GPK** should,

$$\text{for “}m_{1,t} > lc_i\text{” and if } \begin{cases} cl_i[k] \geq lK, & \text{do nothing.} \\ cl_i[k] < lK, & \text{direct a process to force a checkpoint.} \end{cases} \quad (1)$$

Next, we proceed with the other scenario “ $m_{1,t} \leq lc_i$ ”. In this scenario we have $lc_i \geq lK$. Two possible cases need to be examined. But before that, a theorem will be proved.

Theorem 4 For the scenario “ $m_{1,t} \leq lc_i$ ”, if $m_{2,t} < lK$, then $C_{k,z,t} > lK$.

Proof: Since $lc_i \geq lK$ and $m_{2,t} < lK$, there exists a message α with the property that $\alpha t \geq lK$ and $receive(\alpha) \text{ hb } receive(m_1)$. Let the value of αt be $l'K$. Before the event $receive(\alpha)$, the value of the local clock of P_i is less than $l'K$. Obviously $l' \geq l$. At that moment, the value of the local clock lc_i is consequently less than $l'K$. So upon receiving α , the timestamp of α is larger than lc_i at that instant. Since there is no forced checkpoint taken before the event $receive(\alpha)$, according to the discussion of the scenario of $m_{1,t} > lc_i$, we have $cl'_i[k] \geq l'K$, where $cl'_i[k]$ is the value of the local clock of P_k known by P_i at that instant. And if $cl'_i[k] < C_{k,z,t}$, we know that $C_{k,z,t} > l'K$. If $cl'_i[k] \geq C_{k,z,t}$, there is a Z_K -cycle

formed when $C_{k,z}.t \leq l'K$. Hence for both cases, $C_{k,z}.t > l'K$, and thus we can conclude that the argument “ $C_{k,z}.t > l'K$ ” is true. \square

Now we discuss the two possible cases as follows:

- If $m_2.t \geq l'K$, then $C_{k,z}.t > l'K$. This obviously satisfies the design strategy.
- If $m_2.t < l'K$, by applying Theorem 4, the timestamp of $C_{k,z}$ is larger than $l'K$. Hence no forced checkpoint is needed.

From the foregoing discussion, we conclude that protocol **GPK** should,

for “ $m_1.t \leq lc_i$ ”, do nothing. (2)

From Eqs. (1) and (2), we know that $cl_i[k]$ only has to be compared with $l'K$, where l is 0, 1, 2, Hence it is sufficient to consider $CL_i[k]$, where $CL_i[k] = \lfloor cl_i[k] / K \rfloor K$. Note that by applying the basic timestamp management, we have $CL_i[k] \leq lc_i$. In the following section, we describe how to use piggybacked information to implement protocol **GPK** and derive a family of checkpointing protocols from it.

4. IMPLEMENTATION OF THE GENERAL PROTOCOL

To begin, we discuss how to detect Z_K -cycles. As in [9], we need an array $ckpt_i$ of size N , where $ckpt_i[k]$ = the number of checkpoints taken by P_k to P_i 's knowledge. This vector clock is managed in the usual way [17]. When P_i sends a message m , P_i appends to m the current value of $ckpt_i$. Let $m.ckpt$ denote this value. Note that, lc_i and $ckpt_i[i]$ may be different. Variable $ckpt_i[i]$ is used to generate the indices associated with the checkpoints taken by P_i ; this variable takes on the successive values 0, 1, 2, On the other hand, variable lc_i is used to timestamp checkpoints taken by P_i ; due to clock updates, this variable can skip some integer values. Both variables can only increase. Second, we also need the boolean array $taken_i$ used in [9] to cooperate with $ckpt_i$ to detect Z_K -cycles. However, it is managed in a different way. Also, every process has to preserve one more boolean array tc_i of size N . Variables $taken_i$ and tc_i are managed in the following way:

- Variables $taken_i$ and tc_i are initialized to be false.
- When P_i takes a checkpoint, for any $k \neq i$, let $tc_i[k] = true$ ($tc_i[i]$ always remains false).
- Whenever the local clock lc_i becomes $l'K$ where l is some positive integer, for any $k \neq i$, $taken_i[k]$ is set to true if $tc_i[k]$ is true ($taken_i[i]$ always remains false, too).
- When P_i sends a message m , P_i appends to m the current value of $taken_i$ (let $m.taken$ be this value).
- When P_i receives m , P_i updates $taken_i$ in the following way:
 - For any $k \neq i$,
 - if $m.ckpt[k] < ckpt_i[k]$ then skip $m.taken[k]$;
 - if $m.ckpt[k] > ckpt_i[k]$ then if $m.taken[k] = true$, we let $taken_i[k] = true$; else both $taken_i[k]$ and $tc_i[k]$ are set to false;
 - if $m.ckpt[k] = ckpt_i[k]$ then $taken_i[k] = taken_i[k] \vee m.taken[k]$.

With these data structures, the condition to detect Z_K -cycles can be expressed as

$$C_0 \equiv (m_1.ckpt[i] = ckpt_i[i]) \wedge m_1.taken[i].$$

The first part of this condition states that there is a causal path starting after some checkpoint X at P_i and arriving before the next checkpoint of X , while the second part indicates this causal path includes a checkpoint with the timestamp closest to IK and the execution point when lc_i becomes IK . Note that for the special case of $K = 1$, the Z_1 -cycle is just a Z -cycle composed of two causal paths. It can easily be verified that to detect such a Z -cycle, variable tc_i is no longer necessary and the management of variable $taken_i$ can be simplified in the same way as managed in [9].

Now we discuss how to use piggybacked information to express Eqs. (1) and (2). Similarly, we exploit the boolean array $greater_i$ used in [9]. But it has a more generalized meaning: $greater_i[k] = (V_i > CL_i[k])$, where V_i denotes the value of $\lfloor lc_i / K \rfloor K$ for the sake of neatness. This variable $greater_i$ is managed in the following way:

- For any $k \neq i$, $greater_i[k]$ is initialized to be true ($greater_i[i]$ always remains false).
- When P_i sends a message m , P_i appends to m the current value of $greater_i$ (let $m.greater$ be this value).
- When P_i receives m , P_i updates $greater_i$ in the following way:
For any $k \neq i$,
 - if $m.t < V_i$ then skip $m.greater[k]$;
 - if $m.t > V_i$ then let $greater_i[k] = m.greater[k]$;
 - if $m.t = V_i$ then let $greater_i[k] = greater_i[k] \wedge m.greater[k]$.

According to the management of variable $greater_i$, the statement “ $m.greater$ is true” indicates that the *new* $CL_i[k]$ brought by message m is less than $m.t$. Note that the value of $m.t$ equals the value of V_j at the time message m was sent by P_j . Moreover, the condition “ $m.t > lc_i$ ” can deduce that the *original* $CL_i[k]$ preserved by process P_i is less than $m.t$ because $CL_i[k] \leq lc_i$. Hence Eqs. (1) and (2) can be implemented by the condition

$$C_1 \equiv (\exists k : sent_to_i[k] \wedge m_1.greater[k]) \wedge m_1.t > lc_i,$$

where $sent_to_i[k]$ is a boolean array used in [9] and is managed by process P_i in order to know whether P_i has sent a message to P_k since its last checkpoint. Then **GP**K is based on condition C_1 in conjunction with C_0 as follows:

$$C_1 \wedge C_0 \equiv ((\exists k : sent_to_i[k] \wedge m_1.greater[k]) \wedge m_1.t > lc_i) \vee ((m.ckpt[i] = ckpt_i[i]) \wedge m.taken[i]).$$

Here we use the term “ m ” instead of the term “ m_i ” for more generality.

In [9] a family of checkpointing protocols was proposed. Here we also show that our proposed general protocol can give rise to a family of adaptive checkpointing protocols. These protocols are the generalization of those in [9]. First, a new protocol was obtained by discarding condition C_0 and variable $greater_i$. This new protocol is named **FVASK** (**F**ixed V_i **A**fter **S**end with the measure **K**), which is based on the condition

$$C_2 \equiv \exists k : sent_to_i[k] \wedge m_{1,t} > lc_i.$$

Next, we show that **FVASK** is a viable adaptive checkpointing protocol with domino-effect freedom by the following theorem.

Theorem 5 The protocol **FVASK** satisfies the design strategy.

Proof: Consider the situation of the *Z-path* $[m_1, m_2]$ depicted in Fig. 3. If $m_{1,t} > lc_i$, a forced checkpoint will be taken before receiving m_1 , and thus $[m_1, m_2]$ is no longer a *Z-path*. Therefore we only need to consider the scenario “ $m_{1,t} \leq lc_i$ ”. There are two possible cases:

- If $m_{2,t} \geq m_{1,t}$, this satisfies the design strategy.
- If $m_{2,t} < m_{1,t}$, since $lc_i \geq m_{1,t}$, P_i must receive a message α with $\alpha t \geq m_{1,t}$ before *receive*(m_1). Also, upon receiving α , the value of the local clock at that moment is less than αt . This means that condition C_2 is incurred and a forced checkpoint needs to be taken before *receive*(α). This leads to a contradiction.

From the previous discussion, we know that **FVASK** satisfies the design strategy. \square

Another protocol is derived by further weakening condition C_2 . We eliminate the array $sent_to_i$ and get the condition

$$C_3 \equiv m.t > lc_i.$$

This protocol is denoted as **FVIK** (Fixed V_i Interval with the measure K), and is a variant of the lazy coordination addressed in [14]. So the protocol **GPK** can provide a framework that unifies previous works in [7-9, 14].

5. ANALYSES AND COMPARISONS

5.1 Determination of Consistent Global Checkpoints

In [9] Helary *et al.* found an interesting property for the special case $K = 1$ of the timestamp mechanism of the design strategy. It allows an easy timestamp-based determination of consistent global checkpoints. Now we show that this property is also valid in this mechanism for any positive integer K .

Theorem 6 Consider a checkpoint and communication pattern satisfying the timestamp mechanism of the design strategy. For an integer $l \geq 0$, let the global checkpoint $C_G = \{C_{1,x_1}, \dots, C_{N,x_N}\}$, where for any i , C_{i,x_i} is the last checkpoint of P_i with timestamp not larger than lK . Then C_G is consistent.

Proof: By contradiction, suppose C_G is not consistent. By Theorem 2, there must exist two local checkpoints X and Y in C_G such that there is a *Z-path* ζ_1 from X to Y . Let $next(X)$

denote the next checkpoint of X . According to the assumption that the timestamp of every local checkpoint in C_G is less than or equal to IK , we have that $Y.t \leq IK$. Also from the assumption of C_G , we know that $next(X).t > IK$. Based on $Y.t \leq IK$ and the design strategy, we have $X.t < IK$. From the basic timestamp management, since $X.t < IK$ and $next(X).t > IK$, we can conclude that the timestamp of $next(X)$ is obtained from a local checkpoint W that belongs to a process different from the one that has taken X and possesses the property $W.t \geq IK$. So, there is a ζ_2 from W to $next(X)$, as shown in Fig. 5. And then there is also a Z -path $\zeta_2 \cdot \zeta_1$ from W to Y . From the design strategy and $W.t \geq IK$, we get $Y.t > IK$. This leads to a contradiction. \square

This result shows that our adaptive checkpointing protocols can ensure progression of the recovery line, and also provide a flexible tradeoff between the cost of checkpointing coordination and the rollback distance.

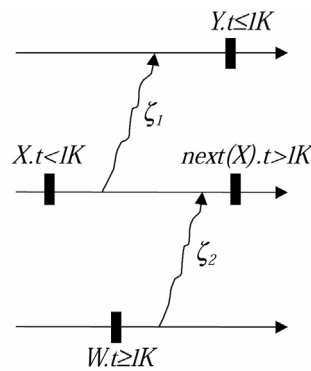


Fig. 5. The scenario of Theorem 6.

5.2 Overhead Analyses

Since the checkpoint overhead of checkpointing protocols depends on the run-time dynamic checkpoint and communication patterns, it is important to analyze and estimate the potential extra overhead from forced checkpoints. Here we first show that the idea of adaptive checkpoint coordination is viable by giving an overhead analysis. Namely, we show that for our proposed adaptive checkpointing protocols, the larger measure K is, the smaller the number of forced checkpoints. Because the checkpoint-inducing condition of an adaptive checkpointing protocol with measure K_1 does not necessarily imply the condition of the same type of protocol with measure K_2 , even though $K_1 < K_2$ (e.g., the condition of **FVI2** does not imply that of **FVI3**), we adopt the worst-case analysis scheme proposed in [14] to estimate overhead between different values of K . Likewise, the approach to worst-case analysis consists of two steps. First, given any basic checkpoint pattern, we construct the worst-case checkpoint and communication pattern. Second, given any system with N processes and measure K , we derive the worst-case *induction ratio*, i.e. #forced checkpoints/#basic checkpoints, as a function of N and K by considering the worst-case checkpoint and communication pattern. Because our checkpointing

protocols are different from the lazy checkpoint coordination scheme of [14] and are more general, we must construct the worst-case pattern in a different way. Given any basic checkpoint pattern and measure K , we construct the worst-case checkpoint and communication pattern pat_o as below.

Let $C_{*,lK}$ denote the earliest checkpoint with a timestamp equivalent to lK among all processes where $l \geq 1$. If $C_{*,lK} = C_{i,lK}$, then for any $j \neq i$, after $C_{i,lK}$ process P_i sends message m_{ij} to process P_j and before receiving m_{ij} , P_j has sent a message m'_{ij} to P_i . The event $receive(m'_{ij})$ in P_i occurred before checkpoint $C_{i,lK}$. Thus the Z -path $[m_{ij}, m'_{ij}]$ forms a Z -cycle. Since this type of Z -cycle has to be broken, a forced checkpoint is taken before receiving message m_{ij} . Fig. 6 shows an example of the worst-case pattern pat_o with $K = 2$.

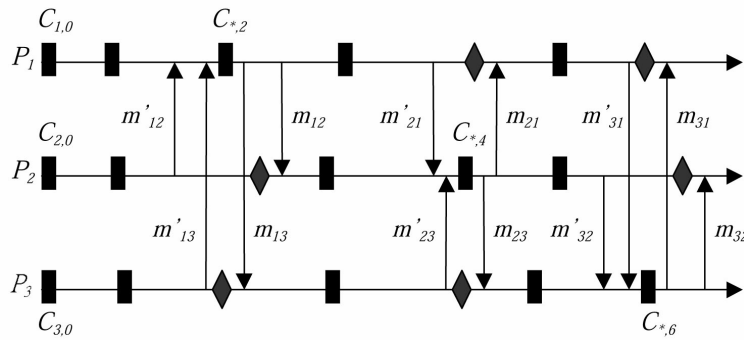


Fig. 6. An example of the worst-case pattern with $K = 2$.

Now we explain why pattern pat_o is the worst case and derive the upper-bound of the induction ratio from pat_o . By the construction of pat_o , since the process possessing checkpoint $C_{*,lK}$ sends a message to every other process which has a value of the local clock less than lK , the sent message will make every other process update its local clock as lK . Hence pattern pat_o always has the earliest $C_{*,lK}$ among all checkpoint and communication patterns, given the basic checkpoint pattern. Since each $C_{*,lK}$ in pat_o also induced the largest possible number $(N - 1)$ of forced checkpoints, the total number of forced checkpoints in pat_o must be the largest, and so we can conclude that given a basic checkpoint pattern, pattern pat_o is the worst-case checkpoint and communication pattern resulting in the largest induction ratio. Moreover, because the induction of any forced checkpoint cannot occur until checkpoint $C_{*,lK}$ is taken and the process having $C_{*,lK}$ must take K consecutive basic checkpoints by itself in order to reach $C_{*,lK}$, we know that at least K basic checkpoints are needed to induce at most $(N - 1)$ forced checkpoints. So the upper-bound of the induction ratio R is,

$$R \leq \frac{N-1}{K}$$

From the foregoing equation, we can conclude that with larger measure K , the number of forced checkpoints is smaller.

5.3 Related Work

Communication-induced checkpointing protocols can be classified into two distinct categories, *index-based* and *model-based* [4]. An index-based protocol associates local checkpoints with sequence numbers similar to Lamport's logical clocks [15] in such a way that checkpoints with the same sequence numbers are forced to be consistent. A model-based protocol does not use a timestamping function, but prevents the formation of special checkpoint and communication patterns in the execution. By comparison, index-based protocols often have fewer forced checkpoints and less control information than model-based ones. Among existing checkpointing protocols with domino-effect freedom, protocols introduced in [7-11] belong to the index-based category and are based on three kinds of indexing strategies. In [18] these strategies are called *normal indexing* [7-9], *lazy indexing* [11] and *lazier indexing* [10]. The latter two can be regarded as optimizations of the first one. On the contrary, protocols proposed in [12, 13] belong to model-based category.

In [19] several simulation experiments were conducted to compare the performance of some index-based and model-based protocols with domino-effect freedom. The authors found that if the communication patterns under study mimic a periodic broadcast, model-based protocols appears to be "eager" in taking forced checkpoints to prevent the formation of *Z-cycles* compared to index-based ones. In [18] a few theoretical results on performance comparisons of some protocols with domino-effect freedom were introduced. The discussions in [18] can be exploited to explain the foregoing simulation result more than those described in [19]. In [19] each process running the simulation has the same rate of taking basic checkpoints such that the sequence number of every process is almost equivalent at the same instant. Also, those parallel applications under study use a common iterative structure to solve a computation intensive problem in which processes change results and then resume. Thus a process often sends a message to some process with a larger or equivalent sequence number, and has received a message from another process in the previous interval. This leads to the checkpoint-inducing condition of a model-based protocol occurring more frequently than that of an index-based one [18]. On the contrary, if a process usually sends a message to another process with a sequence number not larger than its own, then model-based protocols outperform index-based ones for such a pattern [18].

6. CONCLUSIONS

Since extra forced checkpoints are the main factor of run-time overhead, it is desirable to have as small a number of forced checkpoints as possible in practical applications. In this paper, we have presented a family of adaptive checkpointing protocols, for which we can reduce the number of forced checkpoints by defining a larger value of the measure K according to our requirements. Moreover, the proposed **GPK** protocol provides a framework that unifies previous works [7-9, 14]. Most existing checkpointing protocols that solve the problem of domino-effect freedom can be viewed as a particular instance of it. Finally, we showed how to determine consistent global checkpoints for this family of checkpointing protocols, and also gave an overhead analysis to demonstrate that the idea of adaptive checkpoint coordination is viable.

ACKNOWLEDGMENTS

The authors wish to express their sincere thanks to the anonymous referees for their valuable comments. Moreover, this work was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC 89-2213-E-002-114.

REFERENCES

1. Y. M. Wang, A. Lowry, and W. K. Fuchs, "Consistent global checkpoints based on direct dependency tracking," *Information Processing Letters*, Vol. 50, 1994, pp. 223-230.
2. K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transactions on Computing Systems*, Vol. 3, 1985, pp. 63-75.
3. B. Randell, "System structure for software fault-tolerant," *IEEE Transactions on Software Engineering*, Vol. 1, 1975, pp. 220-232.
4. E. N. Elnozahy, L. Alvisi, Y. M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Computing Surveys*, Vol. 34, 2002, pp. 375-408.
5. R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Transactions on Software Engineering*, Vol. 13, 1987, pp. 23-31.
6. B. Janssens and W. K. Fuchs, "Experimental evaluation of multiprocessor cache-based error recovery," in *Proceedings of International Conference on Parallel Processing*, 1991, pp. 505-508.
7. D. Briatico, A. Ciuffoletti, and L. Simoncini, "A distributed domino-effect free recovery algorithm," in *Proceedings of 4th IEEE Symposium on Reliability in Distributed Software and Database Systems*, 1984, pp. 207-215.
8. D. Manivannan and M. Singhal, "A low overhead recovery technique using quasi-synchronous checkpointing," in *Proceedings of 16th IEEE International Conference on Distributed Computing Systems*, 1996, pp. 100-107.
9. A. Mostefaoui, J. M. Helary, R. H. B. Netzer, and M. Raynal, "Communication-based prevention of useless checkpoints in distributed computations," *Distributed Computing*, Vol. 13, 2000, pp. 29-43.
10. R. Baldoni, F. Quaglia, and P. Fornara, "An index-based checkpointing algorithm for autonomous distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, 1999, pp. 181-192.
11. G. M. D. Vieira, I. C. Garcia, and L. E. Buzato, "Systematic analysis of index-based checkpointing algorithms using simulation," in *Proceedings of IX Brazilian Symposium on Fault-Tolerant Computing*, 2001.
12. I. C. Garcia and L. E. Buzato, "Checkpointing using local knowledge about recovery lines," Technical Report, TR-IC-99-22, University of Campinas, Brazil, 1999.
13. F. Quaglia, R. Baldoni, and B. Ciciani, "On the no-Z-cycle property in distributed executions," *Journal of Computer and System Sciences*, Vol. 61, 2000, pp. 400-427.
14. Y. M. Wang and W. F. Fuchs, "Lazy checkpoint coordination for bounding rollback propagation," in *Proceedings of 12th IEEE Symposium on Reliable Distributed Sys-*

- tems*, 1993, pp. 78-85.
15. L. Lamport, "Time, clocks and the ordering of events in a distributed system," *Communications of the ACM*, Vol. 21, 1978, pp. 558-565.
 16. R. H. B. Netzer and J. Xu, "Necessary and sufficient conditions for consistent global snapshots," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, 1995, pp. 165-169.
 17. C. J. Fidge, "Logical time in distributed computing systems," *IEEE Computer*, Vol. 24, 1991, pp. 11-76.
 18. J. Tsai and J. W. Lin, "On characteristics of DEF communication-induced checkpointing protocols," in *Proceedings of Pacific Rim International Symposium on Dependable Computing*, 2002, pp. 29-36.
 19. L. Alvisi, E. Elnozahy, S. Rao, S. A. Husain, and A. D. Mel, "An analysis of communication-induced checkpointing," in *Proceedings of IEEE Fault-Tolerant Computing Symposium*, 1999, pp. 242-249.



Jichiang Tsai (蔡智強) received his B.S. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 1991. Then he started his graduate study at the same university, and received the Ph.D. degree in Electrical Engineering in 1999. Dr. Tsai served as a postdoctoral research fellow in the Institute of Information Science, Academia Sinica, Taipei, Taiwan, from 1999 to 2001. Since 2002, he has been an assistant professor with the Department of Electrical Engineering, National Chung Hsing University, Taichung, Taiwan. His current research interests include fault tolerance, parallel and distributed systems, computer architecture, operating systems and computer networks.



Chi-Yi Lin (林其諳) received the B.S. and Ph.D. degrees in Electrical Engineering from National Taiwan University in 1995 and 2003, respectively. From August-December 2000, he worked in AT&T Labs-Research at New Jersey as a visiting consultant. He is a teaching assistant in the Department of Electrical Engineering, National Taiwan University. His research interests include fault tolerance, distributed and mobile computing systems.



Sy-Yen Kuo (郭斯彦) received the B.S. (1979) in Electrical Engineering from National Taiwan University, the M.S. (1982) in Electrical and Computer Engineering from the University of California at Santa Barbara, and the Ph.D. (1987) in Computer Science from the University of Illinois at Urbana-Champaign. Since 1991 he has been with National Taiwan University, where he is currently a professor and the Chairman of Department of Electrical Engineering. He spent his sabbatical year as a visiting researcher at AT&T Labs-Research, New Jersey from 1999 to 2000. He was the Chairman of the Department of Computer Science and Information Engineering, National Dong Hwa University, Taiwan from 1995 to 1998, a faculty member in the Department of Electrical and Computer Engineering at the University of Arizona from 1988 to 1991, and an engineer at Fairchild Semiconductor and Silvar-Lisco, both in California, from 1982 to 1984. In 1989, he also worked as a summer faculty fellow at Jet Propulsion Laboratory of California Institute of Technology. His current research interests include mobile computing and networks, dependable distributed systems, software reliability, and optical WDM networks. Professor Kuo is an IEEE Fellow. He has published more than 180 papers in journals and conferences. He received the distinguished research award (1997-2005) from the National Science Council, Taiwan. He was also a recipient of the Best Paper Award in the 1996 International Symposium on Software Reliability Engineering, the Best Paper Award in the simulation and test category at the 1986 IEEE/ACM Design Automation Conference (DAC), the National Science Foundation's Research Initiation Award in 1989, and the IEEE/ACM Design Automation Scholarship in 1990 and 1991.