

Short Paper

Applying the Fully-Informed Checkpointing Protocol to the Lazy Indexing Strategy*

JICHIANG TSAI

*Department of Electrical Engineering
National Chung Hsing University
Taichung, 402 Taiwan
E-mail: jichiangt@nchu.edu.tw*

Communication-induced checkpointing (CIC) protocols can be used to prevent domino effect. The fully-informed (FI) protocol proposed in the literature is known as the best CIC protocol so far. Such a protocol is originally designed according to the classical indexing strategy. In this paper, we show how to apply the FI protocol to another improved indexing strategy, called lazy indexing strategy in the context. The lazy-indexing version of such a protocol can reduce number of forced checkpoints for a distributed computation with the lazy indexing strategy. Particularly, the management of a boolean array in FI's control information carried on a message must be modified. Finally, we present a simulation experiment to analyze the impact of such a modification.

Keywords: fault tolerant computing, checkpointing protocols, distributed systems, domino effect, rollback-recovery

1. INTRODUCTION

Checkpointing is an efficient way for a process to recover from a transient failure. In a distributed computation, if every process takes its local checkpoints independently, there is a risk of causing the well-known problem of *domino effect* [1], in which cascading rollbacks in a process will occur during all processes try to rollback to the recovery line. Many protocols have been proposed to selectively take local checkpoints to eliminate the possibility of the domino effect (see the survey paper [2]). One way to avoid the domino effect is to prevent any checkpoint from becoming a *useless checkpoint* [3], *i.e.* a checkpoint that cannot belong to any consistent global checkpoint. Coordinated checkpointing protocols [4, 5] achieve this goal by synchronizing the checkpointing actions of all processes through explicit control messages. In contrast, *communication-induced checkpointing protocols* [6] typically prevent the domino effect by piggybacking control information on application messages. Particularly, besides taking *basic* checkpoints independently, each process can also be asked by a CIC protocol to take extra *forced* checkpoints according to a certain *checkpoint-inducing condition* tested whenever a message is

Received July 26, 2005; accepted November 16, 2005.

Communicated by Sy-Yen Kuo.

* This work was supported by the National Science Council of Taiwan, R.O.C., under grant No. NSC 94-2213-E-005-021.

received. Such a condition is constituted by the piggybacked control information on a message as well as local control variables in a process.

Communication-induced checkpointing (CIC) protocols can be classified into two distinct categories: *index-based* and *model-based* [2]. An index-based protocol associates each local checkpoint with a sequence number in a way that checkpoints with the same sequence number are assured to be consistent; while a model-based protocol is generally more sophisticated and tracks the checkpoint and communication pattern to prevent the formation of certain patterns during the execution. Index-based protocols usually take fewer forced checkpoints and piggyback less control information on messages than model-based ones [7]. Among existing CIC protocols, those proposed in [3, 8-11] belong to the index-based category; whereas those introduced in [12-14] belong to the model-based category.

A remarkable index-based CIC protocol presented in [3] has the strongest checkpoint-inducing condition known so far, and can accomplish an excellent performance, *i.e.* it can force as few checkpoints as possible while still preventing the domino effect. Such a protocol is called the fully-informed (FI) protocol in the context because it exploits as much useful information available from the causal past as possible. In particular, a process collects information about sequence numbers of other processes by control information piggybacked on messages so as to make a checkpointing decision more precisely. Furthermore, index-based CIC protocols must cooperate with their underlying *indexing strategies* to avoid the domino effect. The indexing strategy used by protocols introduced [3, 8, 9] is maintained in the classical way [15] that the sequence number is increased by one each time a basic checkpoint is taken. Such an indexing strategy is called the *classical* indexing strategy in this paper. With this strategy, if a process takes basic checkpoints at a higher rate and consequently have a larger sequence number than other processes, forced checkpoints may be induced when other processes receives messages from this process. To deal with this asymmetry, an improved indexing strategy was proposed in [10]. Such a strategy is called the *lazy* indexing strategy in the context. Its basic concept is that if a process has only received messages with sequence numbers smaller than its own in the current checkpoint interval, it can conclude that its sequence number is ahead and is unnecessary to be increased when the next basic checkpoint is taken.

Although some existent simple CIC protocols can be directly applied to the lazy indexing strategy [10], we discover that the sophisticated FI protocol cannot be directly applied to this strategy. Hence, in this paper, we present how to apply such a protocol to the lazy indexing strategy to decrease number of forced checkpoints for a distributed computation adopting this strategy. In particular, one boolean array in FI's control information piggybacked on messages must be managed in a way adapted from the original one proposed in [3]. At last, we present a simulation study in this paper. The simulation is performed in the typical *point-to-point* computational environment. The influence of the introduced adaptation is discussed.

This paper is structured as six sections. Section 2 defines some technical terms used in the context. In section 3, we describe the concept of the FI protocol, and then how to apply such a protocol to the lazy indexing strategy is shown in the next section. In section 5, we conduct a simulation experiment to analyze our result. Last but not least, we conclude the paper in section 6.

2. PRELIMINARIES

2.1 Z-Paths and Z-Cycles

A distributed computation consists of a finite set P of n processes $\{P_1, P_2, \dots, P_n\}$ that communicate and synchronize only by exchanging messages. We assume that each ordered pair of processes is connected by an asynchronous, reliable, directed logical channel with unpredictable but finite transmission delays, *i.e.* no message will be lost in the channel. Moreover, 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 “*send*(m) to P_j ”, it puts message m into the channel from P_i to P_j . When P_i executes “*receive*(m)”, it is blocked until at least one message directed to P_i has arrived. Then, a message is retrieved from one of its input channels and delivered to P_i . Executions of internal, send and receive statements are modeled by internal, sending and receiving events, respectively [3].

Processes of a distributed computation are *sequential*: each process P_i produces a *sequence* of events. All the events produced by a distributed computation can be partially ordered with Lamport’s well-known *happened-before* relation “ \xrightarrow{hb} ”, defined as follows [15].

Definition 1 The relation “ \xrightarrow{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 $a \xrightarrow{hb} b$;
2. If a is the event *send*(m) and b is the event *receive*(m), then $a \xrightarrow{hb} b$;
3. If $a \xrightarrow{hb} b$ and $b \xrightarrow{hb} c$ then $a \xrightarrow{hb} c$

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 the *process id* and x is the *checkpoint index*. The sequence of events occurring at P_i between $C_{i,x-1}$ and $C_{i,x}$ ($x > 0$) is called a *checkpoint interval* (or *interval* for short), denoted by $I_{i,x}$. Moreover, we assume that each process P_i starts its execution with an initial checkpoint $C_{i,0}$.

In [16], the following notions of *Z-paths* and *Z-cycles* were introduced, and the authors also showed that a checkpoint $C_{i,x}$ cannot be part of a consistent global checkpoint if and only if it is involved in a Z-cycle.

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 , $1 \leq i \leq q - 1$: $receive(m_i) \in I_{k,s} \wedge send(m_{i+1}) \in I_{k,t} \wedge s \leq t$.

Namely, in a Z-path, every message is received in the same or an earlier interval than the succeeding message is sent. In addition, if a Z-path satisfies that its first message is sent by P_i after $C_{i,x}$ and its last message is received by P_j before $C_{j,y}$, 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 one is called a *Z-cycle*. We say that it involves the local checkpoint $C_{i,x}$, and checkpoint $C_{i,x}$ is

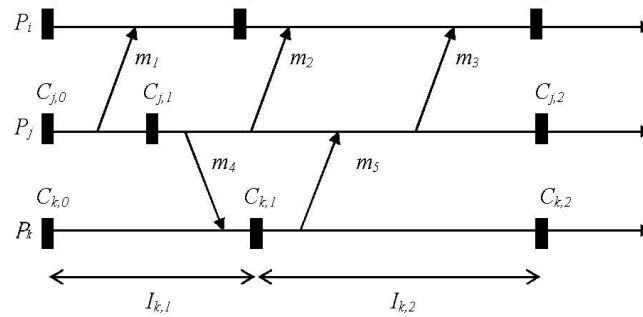


Fig. 1. A checkpoint and communication pattern.

called *useless*. 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 $C_{j,2}$. Checkpoint $C_{k,1}$ is useless because it is involved in a Z-cycle $[m_5, m_4]$. It becomes clear that a CIC protocol can avoid the domino effect by eliminating all Z-cycles with additional forced checkpoints. We say that Z-cycles are *broken*.

A Z-path is *causal* if the receiving event of each message (except the last one) precedes the sending event of the next message in the sequence [16]. A Z-path is *non-causal* if it is not causal. For simplicity, a causal Z-path is also called a *causal path*. As an example, in Fig. 1, Z-path $[m_5, m_3]$ is causal; while Z-path $[m_5, m_2]$ is non-causal. Obviously, a Z-cycle is non-causal. Finally, 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 and ζ' , if their concatenation is also a Z-path, we denote the concatenation as $\zeta \cdot \zeta'$.

2.2 Indexing Strategies

Basically, a common strategy among index-based CIC protocols to eliminate any Z-cycle is to guarantee that sequence numbers for checkpoints always *increase* along a Z-path, as stated in the following theorem.

Theorem 1 Let each checkpoint C be associated with a sequence number $C.sn$. If for every pair of checkpoints $C_{i,x}$ and $C_{j,y}$ with a Z-path from $C_{i,x}$ to $C_{j,y}$, we have $C_{i,x}.sn < C_{j,y}.sn$, then there is no Z-cycle [9, 17].

Sequence numbers of the underlying indexing strategy used by protocols in [3, 8, 9] is maintained in the following classical way [15]:

- Each process P_i manages a sequence number sn_i (initialized to be zero).
- Before taking a basic checkpoint, P_i increases sn_i by 1 and assigns the new value to be the sequence number of this checkpoint.
- Upon sending a message m , P_i indexes m with its current sn_i (let $m.sn$ denote the sequence number of m).
- Upon receiving a message m , P_i updates sn_i to $\max(sn_i, m.sn)$.

Obviously, the classical indexing strategy can only assure that for two 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 $C_{j,y}$ where $i \neq j$, sequence numbers of the two checkpoints satisfy the assumption of Theorem 1. To achieve that for every Z-path, the indexing strategy needs the help of CIC protocols on top of it. For example, a protocol proposed in [8] is based on the “ $(m.sn > sn_i)$ ” condition. Namely, it directs process P_i to force a checkpoint as receiving a message m with a sequence number larger than P_i 's current one. Moreover, another protocol in [3] induces a forced checkpoint whenever it encounters an “ $m.sn > sn_i$ ” condition that is not separated in a different interval from its previous message-sending event.

For the classical indexing strategy, the sequence number is increased by one each time a basic checkpoint is taken. Hence, if a process takes basic checkpoints at a higher rate and consequently have a larger sequence number than other processes, forced checkpoints may be induced when other processes receive a message from such a process because existent index-based protocols are all derived according to the the “ $(m.sn > sn_i)$ ” condition. To deal with this asymmetry, the *lazy indexing* strategy was introduced in [10]. Its basic concept is stated as follows. If a process P_i has only received messages with sequence numbers smaller than its own, P_i can deduce that its sequence number is ahead and does not need to be increased when the next basic checkpoint is taken. On the other hand, if P_i has received one message with the same sequence number with its current one, P_i must increase its sequence number to assure consistency as it takes the next basic checkpoint. Moreover, when P_i receives a message with a larger sequence number than it, P_i updates its sequence number to the one of the message and also increases its sequence number when it takes the next basic checkpoint since P_i has received a message with a sequence number equal to its updated one [10]. Examples of these scenarios are depicted in Fig. 2.

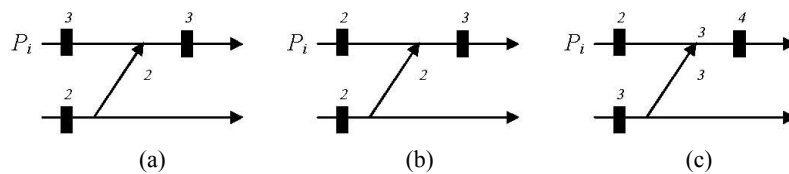


Fig. 2. (a) P_i does not increase its sequence number; (b) P_i increases its sequence number; (c) P_i updates and increases its sequence number.

To implement such a strategy, a process P_i only needs to maintain a flag, $increment_i$, which is set *false* whenever a checkpoint (basic or forced) is taken and becomes *true* as a message with the same or a larger sequence number than P_i arrives. It is clear that for the lazy indexing strategy, a *trivial* causal path, which means a causal path from a checkpoint of a process to a later one of the same process, may not be consistent with the assumption of Theorem 1. But this does not lead to incorrectness since a Z-cycle is never a trivial causal path. Similarly, the lazy indexing strategy must cooperate with CIC protocols to prevent a pattern from the domino effect. The two simple protocols mentioned previously can be directly applied to such a strategy [10].

3. THE FULLY-INFORMED CHECKPOINTING PROTOCOL

The fully-informed CIC protocol was also proposed in [3]. By gathering more information available from the causal past, it can achieve a better performance. Its checkpoint-inducing condition is expressed as below [3]:

$$((\exists k : sent_to_i[k] \wedge m.greater[k]) \wedge m.sn > sn_i) \vee (m.ckpt[i] = ckpt_i[i] \wedge m.taken[i]).$$

The former part “ $(\exists k : sent_to_i[k] \wedge m.greater[k]) \wedge m.sn > sn_i$ ” of the previous condition means a non-causal Z-path that contains only two messages and may not satisfy the assumption of Theorem 1. Thus a forced checkpoint is required to break such a Z-path. Array $sent_to_i[k]$ is an n -size boolean array managed by a process P_i to know whether P_i has sent a message to another process P_k after its last checkpoint. The other array $greater_i$ is also a boolean array of size n with the meaning $greater_i[k] \equiv (sn_i > cl_i[k])$, where $cl_i[k]$ denotes the value of P_k 's sequence number as known by P_i . Note that array cl_i for a process P_i is actually not used by the FI protocol since it is replaced by array $greater_i$ [3]. We mention this array in the context to make some points easy to understand. When P_i sends a message m , P_i appends to m the current value of $greater_i$. Let $m.greater$ denote this value. Upon receiving a message m , “ $m.greater[k] = true$ ” indicates that the new P_k 's sequence number brought by message m is less than $m.sn$. Moreover, “ $m.sn > sn_i$ ” implies that the original P_k 's sequence number preserved by P_i is less than $m.sn$ since we always have $cl_i[k] \leq sn_i$ according to the indexing strategy. Thus the conjunction of these two terms indicates that this non-causal Z-path may not satisfy the assumption of Theorem 1 [3]. For example, in Fig. 3 (a), we may have $C_{k,z}.sn \leq C_{j,y}.sn$ for the non-causal Z-path $[m, m']$ from P_j to P_k through P_i . Note that the causal path μ in the figure brings information about the sequence number of P_k to P_j before m is sent.

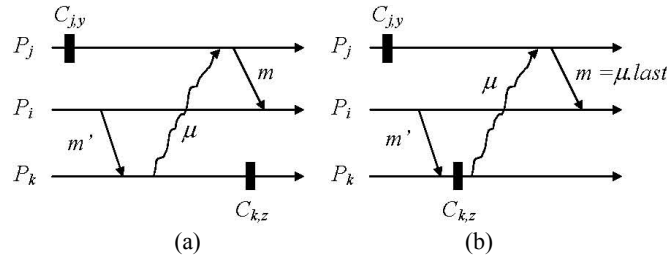


Fig. 3. Situations of the two parts of the FI protocol's condition.

On the other hand, the latter part “ $m.ckpt[i] = ckpt_i[i] \wedge m.taken[i]$ ” of the FI protocol's condition is used to detect a Z_M -cycle, which is a Z-cycle and only the concatenation of a causal path and a single message. An example of Z_M -cycles is depicted in Fig. 3 (b), where the Z_M -cycle from P_k to P_k is constituted by the causal path μ and the message m' . Note that in Fig. 3 (b), m denotes the last message of μ and is sent by P_j . Since a Z_M -cycle is a Z-cycle, it has to be broken by a forced checkpoint. Array $ckpt_i$ is an integer array 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 [18]. The other array $taken_i$ is as well an

n -size boolean array, where $taken_i[k] = true$ in the sense that there is a causal path from the last checkpoint of P_k known by P_i to the next checkpoint of P_i and this causal path includes a checkpoint. When P_i sends a message m , P_i also appends to m the current values of $ckpt_i$ and $taken_i$. Here, $m.ckpt$ and $m.taken$ denote these two values, respectively. So the first term of the latter part means that there is a causal path from a checkpoint of P_i to the succeeding one; while the other term indicates this causal path includes a checkpoint. The conjunction of such two terms exactly implies the existence of a $Z_{M,T}$ -cycle [3].

4. TO THE LAZY INDEXING STRATEGY

For the lazy indexing strategy, the sequence number of a process may not be increased by one when a basic checkpoint is taken. Thus the FI protocol can not be applied to such a strategy directly. We should modify the original management of the boolean *greater* array. Now reconsider the non-causal Z-path $[m, m']$ in Fig. 3 (a). With the lazy indexing strategy, P_k may not increase its sequence number when checkpoint $C_{k,z}$ is taken. This means that $C_{k,z}.sn$ may be equal to the value of sn_k at the moment μ_{first} is sent. Namely, when message m is sent by P_j , we may have $cl_j[k] = C_{k,z}.sn$. So upon receiving m , even though $m.greater[k] = false$, which implies $m.sn = cl_j[k]$, P_i still can not guarantee that $[m, m']$ satisfies the assumption of Theorem 1, namely $C_{k,z}.sn > C_{j,y}.sn$, because we may have $C_{k,z}.sn = cl_j[k] = m.sn = C_{j,y}.sn$. From the previous discussion, we know that $greater_k[k]$, which indicates if $sn_k > cl_k[k]$, can not always be *false* as usual. It must reveal the fact that the sequence number of P_k 's next basic checkpoint may be the same with the current one. As long as flag $increment_k$ remains *false*, i.e. P_k 's sequence number is not yet definitely going to be increased when the next basic checkpoint is taken, $greater_k[k]$ must be *true*. Hence, the management for the boolean *greater_i* array of a process P_i is modified as below:

- When P_i takes a checkpoint, $greater_i[i]$ is set *true*. Moreover, if sn_i is increased by one at this time, $greater_i[k] = true, \forall k \neq i$.
- Upon sending a message m , P_i appends to m the current value of $greater_i$ (let $m.greater$ be this value).
- Upon receiving a message m , P_i updates $greater_i$ in the following way:
 - if $m.sn > sn_i$, $greater_i[i] = false$, and $greater_i[k] = m.greater[k], \forall k \neq i$.
 - if $m.sn > sn_i$, $greater_i[i] = false$, and $greater_i[k] = greater_i[k] \wedge m.greater[k], \forall k \neq i$.

With the foregoing new management for array *greater*, the FI protocol can be applied to the lazy indexing strategy in the sense that any useless checkpoint can be avoided. The lazy indexing counterpart of the FI protocol is called the lazy-FI protocol in the context.

Directly from the results of [3], we know that the FI protocol's condition implies the " $(m.sn > sn_i)$ " condition. In the following theorem, we also demonstrate that the lazy-FI protocol's condition still implies the " $(m.sn > sn_i)$ " condition although the boolean *greater* array is managed in a different way. In [19], formal proofs are proposed to show that comparisons of lazy indexing protocols with a condition implying the " $(m.sn > sn_i)$ " condition can be directly based on their conditions.

Theorem 2 The lazy-FI protocol's condition implies the “ $(m.sn > sn_i)$ ” condition.

Proof: Since the former part of the lazy-FI protocol's condition obviously implies the “ $(m.sn > sn_i)$ ” condition, it is sufficient to prove this theorem by showing that any Z_M -cycle encountered by the lazy-FI protocol also implies such a condition.

Now reconsider the Z_M -cycle depicted in Fig. 3 (b). In the figure, such a Z_M -cycle is represented by $\mu \cdot [m']$ and message m denotes the last message of μ . If $m'.sn \geq sn_k$ when m' is delivered to P_k , we have that $C_{k,z}.sn$ is at least $m'.sn + 1$ from the management of the lazy indexing strategy. On the other hand, if $m'.sn < sn_k$ at the moment m' is delivered to P_k , we also have $C_{k,z}.sn > m'.sn$ because $C_{k,z}.sn \geq sn_k$. Hence, we know $m.sn > m'.sn$ since $m.sn \geq C_{k,z}.sn > m'.sn$.

Now by contradiction, assume $m.sn \leq sn_i$ when m is delivered to P_i . Let $sn_i = t$ at that time, and thus $m.sn \leq t$. We also have $m'.sn < t$ since $m.sn > m'.sn$. This indicates that there exists a message α with $\alpha.sn = t$, received by P_i between $send(m')$ and $receive(m)$. Furthermore, at the moment α is delivered to P_i , the term “ $\alpha.sn > sn_i$ ” holds. Because $send(m')$ and $receive(m)$ belong to the same interval, the value of $\alpha.greater[k]$ must be *false*; otherwise the lazy-FI protocol would direct P_i to take a forced checkpoint before α is received since the former part of its condition holds. Such a result indicates that there is a causal path ν that brings information about the sequence number of P_k to the sender of α before α is sent. Furthermore, since $\alpha.sn = t$ and $\alpha.greater[k] = false$, as the first message of ν is sent, we can assume that P_k has $sn_k = t$ and $greater_k[k] = false$ without loss of generality. This is because a process P_i can make its $greater_i[k]$ equal to *false* only when it receives a message m with $m.greater[k] = false$. The fact “ $greater_k[k] = false$ ” also means $increment_k = true$ at the time $\nu.first$ is sent. Clearly, $\nu.first$ must be sent before $C_{k,z}$; otherwise $\nu \cdot [\alpha] \cdot [m']$ would be a Z_M -cycle and thus a forced checkpoint would be taken before $receive(\alpha)$. Therefore, $C_{k,z}.sn$ is larger than t because $increment_k$ has already been set *true*. This leads to a contradiction because $m.sn \geq C_{k,z}.sn > t$. So the term “ $m.sn > sn_i$ ” holds for this Z_M -cycle as well. \square

5. A SIMULATION STUDY

In the simulation study, we adopt the typical point-to-point computational environment, where a complete network is assumed. Every process first checks if any message is waited to be received. If so, it will manage this message; otherwise, it will either execute a *send* operation with probability 0.1, or an *internal* operation with probability 0.9. The time to execute an operation in a process is exponentially distributed with mean value equal to 1 time unit, and the message propagation time is exponentially distributed with mean value equal to 10 time units. Moreover, it is considered the average of 10 measurements, running with different seeds, for an experiment point. Every measurement is taken by the execution of all considered protocols under the same checkpoint and communication pattern. A simulation run contains 1,000 message deliveries per process on average. Here, we consider the effectiveness of the boolean *greater* array, to see approximately how many forced checkpoints can be eliminated with its help. Thus we count the ratio of the situation “ $m.greater[k] = false$ ” for all non-causal Z -paths $[m, m']$ from a process P_j to a different process P_k via another process P_i and satisfying $m.sn > sn_i$.

Finally, a basic checkpoint is taken every 20 operations for all processes.

Fig. 4 depicts the simulation results, for $4 \leq n \leq 16$, where n denotes the number of processes. We can see that due to the new management of the boolean *greater* array for the lazy indexing strategy, where *greater*_{*i*} must remain *true* until *increment*_{*i*} becomes *true* for a process P_i , the measured ratio of the lazy-FI protocol is smaller. However, it can still take effect to eliminate number of forced checkpoints.

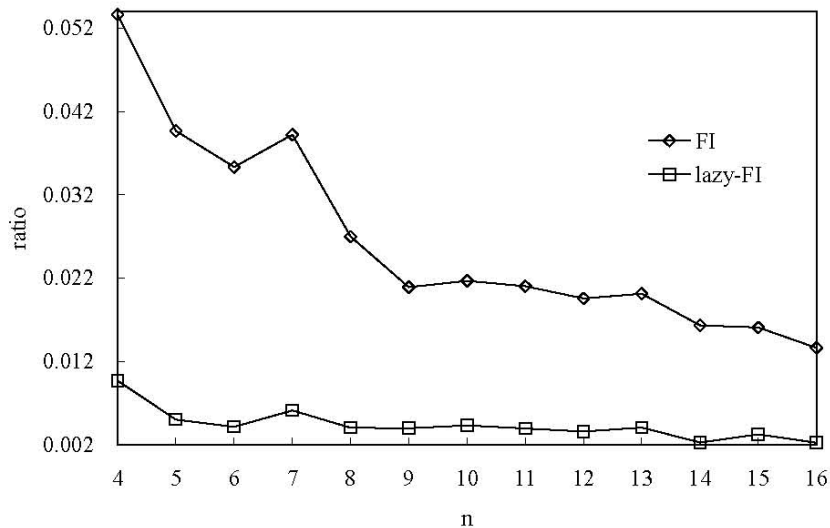


Fig. 4. Simulation results.

6. CONCLUSIONS

In this paper, we showed how to apply the FI protocol to the lazy indexing strategy. The management for array *greater* needs to be modified to reflect the fact that the sequence number of a process may not be increased as a basic checkpoint is taken. Moreover, we presented a simulation study to discuss the impact of our modification. We found that the optimization adopted by the FI protocol applied to the lazy indexing strategy does not work as efficient as applied to classical one. But we can still employ it to reduce number of forced checkpoints for a distributed computation running with the lazy indexing strategy.

REFERENCES

1. B. Randell, "System structure for software fault-tolerant," *IEEE Transactions on Software Engineering*, Vol. 1, 1975, pp. 220-232.
2. 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.

3. 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.
4. 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.
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. L. Alvisi, E. Elnozahy, S. Rao, S. A. Husain, and A. De Mel, "An analysis of communication-induced check-pointing," in *Proceedings of IEEE Fault-Tolerant Computing Symposium*, 1999, pp. 242-249.
8. D. Briatico, A. Ciuffoletti, and L. Simoncini, "A distributed domino-effect free recovery algorithm," in *Proceedings of the 4th IEEE Symposium on Reliability in Distributed Software and Database Systems*, 1984, pp. 207-215.
9. D. Manivannan and M. Singhal, "A low overhead recovery technique using quasi-synchronous checkpointing," in *Proceedings of the 16th IEEE International Conference on Distributed Computing Systems*, 1996, pp. 100-107.
10. 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, pp. 31-41.
11. 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.
12. I. C. Garcia and L. E. Buzato, "Checkpointing using local knowledge about recovery lines," Technical Report No. 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. D. L. Russell, "State restoration in systems of communicating processes," *IEEE Transactions on Software Engineering*, Vol. 6, 1980, pp. 183-194.
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. J. M. Helary, A. Mostefaoui, and M. Raynal, "Virtual precedence in asynchronous systems: concept and applications," in *Proceedings of the 11th International Workshop on Distributed Algorithms*, 1997, pp. 170-184.
18. C. J. Fidge, "Logical time in distributed computing systems," *IEEE Computer*, Vol. 24, 1991, pp. 11-76.
19. 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.

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. He served as a postdoctoral research fellow in the Institute of Information Science, Academia Sinica, Taipei, Taiwan from 1999 to 2001. In 2002, he joined the Department of Electrical Engineering, National Chung Hsing University, Taichung, Taiwan, as an assistant professor, and then was promoted to associate professor in 2005. His current research interests include fault tolerance, parallel and distributed systems, embedded systems, security and real-time operating systems.