Checkpointing in Hybrid Distributed Systems

Jiannong Cao[1] Yifeng Chen[1,2] Kang Zhang[3] Yanxiang He[2]
[1]*Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*
[2]*School of Computing, Wuhan University, Wuhan, Hubei 430072, China*
[3]*Department of Computer Science, University of Texas at Dallas, Richardson, TX 75083-0688, USA*
csjcao@comp.polyu.edu.hk, yfchenco@sina.com, kzhang@utdallas.edu, yxhe@whu.edu.cn

Abstract

To provide fault tolerance to computer systems suffering from transient faults, checkpointing and rollback recovery is one of the widely-used techniques. Among others, two primary checkpointing schemes have been proposed: independent and coordinated schemes. However, most existing works address only the need of employing a single checkpointing and rollback recovery scheme to a target system. In this paper, issues are discussed and a new algorithm is developed to address the need of integrating independent and coordinated checkpointing schemes for applications running in a hybrid distributed environment containing multiple heterogeneous subsystems. The required changes to the original checkpointing schemes for each subsystem and the overall prevented unnecessary rollbacks for the integrated system are presented. Also described is an algorithm for collecting garbage checkpoints in the combined hybrid system.

1. Introduction

Systems may fail from time to time, despite how well they are designed. One of the widely used techniques that allow systems to progress in spite of failure is checkpointing and rollback-recovery [1]. The basic idea is to periodically record the system state as a *checkpoint* during normal system operation and, upon detection of faults, to restore one of the checkpoints and restart the system from there.

Consider a distributed system where processes communicate only through messages. Message passing introduces dependency between the involved processes. Consequently, rollback of one failed process may force rollback of the other processes in order to maintain consistency of the system. In the worst case, consistency requirement may force the system to rollback to the initial state of the system, losing all the work performed before a failure. This uncontrolled propagation of rollback is termed *domino effect* [21]. Thus the main problem in rolling back a distributed system is to keep track of the interactions that cause dependencies among processes and to derive a consistent global checkpoint set (i.e., the *recovery line*), each from one process, to be restored in the event of failure. A system state is said to be *consistent*, if in which no message is recorded in the state of its receiver but not recorded in the state of its sender [9,13].

Two primary approaches have been used for taking checkpoints in distributed systems: *independent* and *coordinated*. Independent checkpointing [3,18,27] allows the participating processes to take checkpoints without any coordination and thus has minimum overhead in the normal execution phase. However, when faults occur, it does not guarantee to have a consistent system state except the initial state. Therefore, most works in independent checkpointing are dedicated to avoiding the undesirable domino effect, finding the latest recovery line, and collecting the useless garbage checkpoints. One approach to preventing the domino effect is to force each process to take checkpoints based on information piggybacked on the application messages received from other processes, which is generally known as communication-induced checkpointing [10,16]. An alternative is coordinated checkpointing [9,14,20] in which all participating processes are coordinated to take a consistent global checkpoint at the expense of a relatively higher overhead during normal operation.

Most existing works [2, 4, 6, 12, 13, 15, 17, 23, 24, 27] are based on the assumption that a single rollback recovery scheme is enough to be applied to the entire system. However, with the advent of large-scale, global computing systems, such those for Grid computing and Web services [8, 11], a computational system may contain many geographically dispersed heterogeneous subsystems for large-scale resource sharing and problem solving. Each subsystem may autonomously contain many processes performing complex operations to collectively achieve a single required function. In such a system, it is inappropriate to have a single rollback recovery scheme for the whole system as each subsystem may require different rollback recovery scheme due to the functional nature of the subsystem.

Since subsystems within the same heterogeneous distributed system may interact through messages which result in dependencies between the subsystems, a collection of consistent local subsystem checkpoint sets does not necessarily produce a consistent system-wide global checkpoint. If the subsystems are simply connected without any modification to their underlying checkpointing schemes, many unnecessary rollbacks will be performed when constructing a consistent global checkpoint. There is therefore a need to have a global checkpointing algorithm that prevents unnecessary rollbacks in reaching a consistent global checkpoint. In this paper, we present a global hybrid checkpointing algorithm that allows individual subsystems to retain their own checkpointing approaches (either independent or coordinated) as much as possible but avoids unnecessary rollbacks whenever the most recent consistent system-wide global checkpoint set is to be restored due to failure of any underlying subsystem. We also present an algorithm for collecting garbage checkpoints in the combined hybrid system.

The remainder of this paper is organized as follows. In Section 2, we state the checkpointing problem in hybrid systems and describe the system model. Section 3 presents a checkpointing scheme for hybrid systems as well as a garbage collection algorithm for reclaiming obsolete checkpoint sets of coordinated checkpointing subsystems in the combined hybrid system. Finally, Section 4 concludes the paper with a discussion of our future work.

2. System Model

2.1. Problem Statement

A hybrid distributed system contains a certain number of subsystems that cooperate to execute a distributed program. Each subsystem, at the lower level, contains a collection of processes. The communication between subsystems and between processes in a subsystem is through message passing. A message either introduces inter-process dependency or inter-subsystem dependency, depending on whether the message is transmitted inside a subsystem or across the boundaries of two subsystems. Subsystems are heterogeneous in the sense that each subsystem can autonomously decide on choosing a suitable checkpointing protocol based on the nature of the target subsystem. In general, independent checkpointing usually finds its application in non-critical and operation repeatable applications such as scientific calculation and corporate administration systems. Coordinated checkpointing, on the other hand, finds its strength in mission critical applications such as a live-supporting system in a hospital and non-idempotent applications such as a ticketing system. Our problem is how to combine different checkpointing subsystems (called *subsystems* hereafter) into a new system (called *hybrid system*), ensuring that individual subsystems are allowed to retain their own checkpointing protocols as much as possible while avoiding unnecessary rollbacks resulted from inter-subsystem dependency. Also, overhead should be reasonable when the most recent consistent system-wide global checkpoint set is to be restored due to failure of any underlying subsystem.

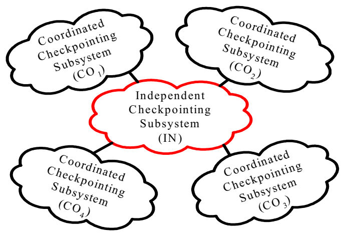
2.2. Model of a Hybrid System

In a hybrid system that contains multiple heterogeneous subsystems, the complexity of combining multiple underlying checkpointing protocols depends on the nature of these subsystems. Combining multiple independent checkpointing subsystems, for instance, is relatively easy. We can simply connect them together and regard the resulting system as a larger independent checkpointing system without affecting the underlying independent checkpointing schemes.

On the other hand, connecting coordinated checkpointing subsystems together is also relatively simple. One approach is to simply connect them together, and treat each coordinated subsystem as a single process in the resulting hybrid system. The scope of taking coordinated checkpoints in each local subsystem remains the same. Then rollback-recovery algorithms for independent checkpointing systems can be applied to the hybrid system to find the system-wide recovery line. This approach minimizes the added runtime overhead for taking checkpoints but it requires the underlying subsystems to keep multiple checkpoint sets. Domino effect may appear and the nature of the coordinated checkpointing schemes employed by the subsystems may also be offended. An alternative approach is to treat the entire integrated system as a single system and the scope of coordinated checkpointing extends to the entire system. This approach adds serious overhead to the target system but guarantees the system to be free from domino effect.

Connecting coordinated checkpointing subsystems with independent checkpointing subsystems is, however, much more complex. Any approach described above will disturb either of the subsystems. For example, simply connecting the subsystems will seriously reduce the recoverability of the coordinated checkpointing subsystems while treating the integrated system as a higher-level coordinated system will lead to a serious performance degradation of the independent checkpointing subsystems.

In this paper, for simplicity and without lose of generality, we only consider the case where multiple coordinated checkpointing subsystems are connected with a single independent checkpointing subsystem. This is motivated by the above observations that multiple independent checkpointing subsystems are relatively easy to be integrated into one larger independent subsystem and several tightly-coupled coordinated checkpointing subsystems can be first integrated into one coordinated subsystem. After this generalization, all the remaining coordinated subsystems are loosely-coupled, and the messages exchanged between them are not very heavy. In addition, we make a further assumption that the generalized coordinated checkpointing subsystems do not communicate with each other directly. Since the current checkpoint set of each generalized coordinated checkpointing subsystem always forms a consistent global state, they can be regarded as a single process in the resulting hybrid system. With this notion in mind, a hybrid system can be regarded as a meta-level independent checkpointing system, with the processes of independent subsystem and the generalized coordinated subsystems at the top level of the hierarchy. Figure 1 illustrates the hybrid system model. For this special model, our goal is to ensure to as much degree as possible, that in the combined hybrid system (i) a rollback in the independent checkpointing subsystem will not lead to a rollback in a coordinated checkpointing subsystem, and (ii) a rollback in a coordinated checkpointing subsystem will not be propagated to another one through the independent checkpointing subsystem.



Coordinated Checkpointing Subsystem (CO1)
Coordinated Checkpointing Subsystem (CO2)
Independent Checkpointing Subsystem (IN)
Coordinated Checkpointing Subsystem (CO4)
Coordinated Checkpointing Subsystem (CO3)

Figure 1. A hybrid system

2.3. Definitions and Notations

In the hybrid system, we use **IN** to represent the independent checkpointing subsystem and **CO** to represent one of the coordinated checkpointing subsystems. Each participating **CO** is viewed as a process and its global checkpoint as a checkpoint. Due to this fact, we use checkpoint set and checkpoint interchangeably for **CO**s in the context. With this generalization, we introduce the following terminologies. A *local checkpoint* is a recorded state (snapshot) of a process. A *global checkpoint* is a set of local checkpoints, one from each process. It is *consistent* if it forms a consistent system state. A *generalized checkpoint*, $C_{i,x}$, is either the *xth* local checkpoint of process P_i ($0 \le i < m$, where m is the number of processes in **IN**) in **IN**, or the *xth* consistent global checkpoint in $\mathbf{CO_i}$ ($i \ge m$). We use $I_{i,x}$ to denote the checkpointing interval between $C_{i,x}$ and $C_{i,x+1}$. A *system-wide* checkpoint is a set of generalized checkpoints, one from each of **CO**s and the processes in **IN**.

A *direct dependency* of generalized checkpoint B on another generalized checkpoint A, denoted by $A \rightarrow B$, implies that a rollback to A requires a rollback to B in order to keep the system consistent. A generalized checkpoint B *transitively depends* on another generalized checkpoint A, denoted by $A \rightarrow^{*} B$, *iff* $A \rightarrow B$ or there exists at least one C such that $A \rightarrow C \rightarrow \ldots \rightarrow B$. Note that the notion of transitive dependency defined in this paper is distinct from that of *zigzag-path* defined in [19]. It is easy to verify, however, that if $C_{i,x} \rightarrow^{*} C_{j,y}$, then there is $C_{i,x} \rightarrow^{z} C_{j,y+1}$, and vice versa, where $A \rightarrow^{z} B$ means a zigzag-path from A to B. A system-wide checkpoint is *consistent* if (i) The set of local checkpoints in **IN** is consistent, forming a consistent global checkpoint; (ii) For any $C_{i,x}$ of *ith* process in **IN** and $C_{j,y}$ in $\mathbf{CO_j}$, if $C_{i,x} \rightarrow C_{j,y}$ then $C_{i,x+1}$ (if exists) must be part of a consistent global checkpoint in **IN**, and (iii) For any $C_{j,y}$ in $\mathbf{CO_j}$ and $C_{k,z}$ in $\mathbf{CO_k}$ ($j \ne k$), neither $C_{j,y} \rightarrow^{*} C_{k,z}$, nor $C_{k,z} \rightarrow^{*} C_{j,y}$. Note that the given conditions are sufficient but not necessary.

For dependency tracking in a hybrid system, we generalize the concept of the *checkpoint graph* [24]. A *generalized checkpoint graph* is a graph whose nodes represent generalized checkpoints and arcs represent dependencies among the generalized checkpoints. It can be constructed according to the following rules: (i) Each generalized checkpoint $C_{i,x}$ is drawn as a node; (ii) An arc $C_{i,x} \rightarrow C_{i,x+1}$ is drawn if $C_{i,x+1}$ exists; and (iii) An arc $C_{i,x} \rightarrow C_{j,y}$ is drawn if $i \ne j$ and P_i sends at least one message in $I_{i,x}$ directly to P_j which receives it in $I_{j,y}$. Figure 2 shows an example of the generalized checkpointing graph representing the checkpoint dependency relationship of a hybrid system formed by connecting a **CO** subsystem and an **IN** subsystem.

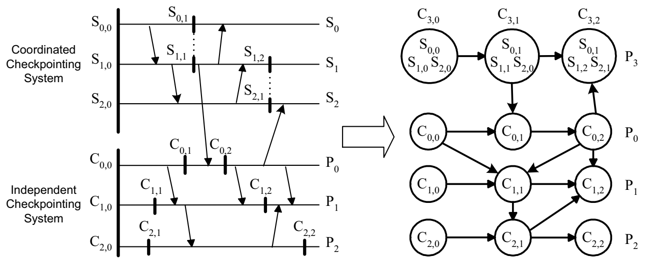


Figure 2. A generalized checkpointing graph

3. Checkpointing in Hybrid Systems

3.1. The Checkpointing Algorithm

After starting with a consistent global checkpoint set, an application message exchanged between subsystems **IN** and **CO** may introduce one of the following direct dependency arcs in the generalized checkpoint graph: (i) Arc from **IN** to **CO**: a process in **IN** sends a message M to **CO**; or (ii) Arc from **CO** to **IN**: **CO** sends a message M to a process in **IN**. To address the dependency problem introduced in these two cases, our checkpointing algorithm consists of two parts stated as two checkpointing rules.

***Rule 1**: **IN** takes a new coordinated checkpoint set if it sends a message to any **CO**.* Namely, **IN** takes a new coordinated checkpoint set immediately after sending M to any **CO** using a developed algorithm, e.g., in [5,13,15]. Since we assume that this kind of message flow is kept to a minimum, we can afford to take a new coordinated checkpoint set in **IN** whenever an application message is sent from **IN** to **CO**. An example situation is shown in Figure 3.

The rule implies that (i) Sending M and taking a new coordinated checkpoint of **IN** must be an atomic action, which can be implemented as shown in Part 1 of Algorithm 1; (ii) **IN** itself becomes a hybrid checkpointing system; and (iii) Taking a new coordinated checkpoint of **IN** is within the scope of **IN** and involves no other **CO**.

***Rule 2**: P_i of **IN** takes a new checkpoint before receiving message from **CO** if P_i has sent a message after taking the current checkpoint.* That is, P_i of **IN** takes a new independent checkpoint before receiving M from any **CO** (or logically "delay" the receiving of M) if there is an outgoing arc from the current checkpoint of P_i. This can be accomplished by: (i) making P_i temporarily keep the message in the stable storage until it takes the new checkpoint, or (ii) using a control message (as opposed to application message) to inform P_i to take a new checkpoint and then wait for the checkpoint taken confirmation from P_i before sending *M*. An example situation of Rule 2 is shown in Figure 4, and an example implementation is illustrated in Part 2 of Algorithm 1.

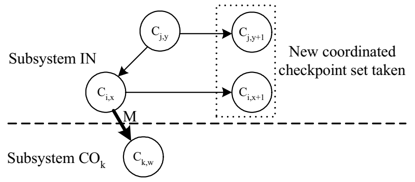


Figure 3. IN takes a new coordinated checkpoint immediately after IN sends *M* to any CO

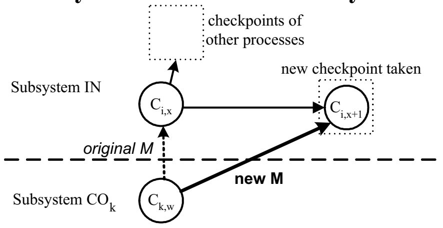


Figure 4. P_i in IN takes $C_{i,x+1}$ before receiving *M* from CO if P_i has sent any message during current checkpointing interval

Algorithm 1 The checkpointing algorithm (run by **IN**)

```
Part 1: Implementation of Rule 1
1: If M is destined to any CO_k {
2:     Lock the system state of IN;
3:     S = current system state of IN + "M is sent";
4:     Write S to stable storage;
5:     Send M to CO;
6:     Record S as the new coordinated checkpoint set;
7:     Unlock the system state of IN; }
Part 2: Implementation of Rule 2
1: If M is sent from any CO_k and P_i has sent any message during
     current checkpointing interval before the arrival of M
   { Take a new checkpoint; }
2: Record the receipt of M;
3: Process M;
```

We now analyze and discuss the implication of the checkpointing algorithm for the combined hybrid system. The major features of the algorithm can be summarized as follows: (i) It introduces relatively low extra overhead to the hybrid system and thus does not significantly hurt the performance of the overall system; (ii) It requires only few changes to coordinated checkpointing subsystems in the combined hybrid system; and (iii) It does not need to shutdown participating coordinated checkpointing subsystems when applying the algorithm. In the algorithm, Rule 1 eliminates the possibility of rollback in any $\mathbf{CO_i}$ originated from **IN**. Thus the reliability of $\mathbf{CO_i}$ after connecting to **IN** is maintained. Rule 2 removes the dependency that would otherwise exist between $\mathbf{CO_i}$ and $\mathbf{CO_j}$ if P_k in **IN** sends a message to $\mathbf{CO_j}$ and then receives a message from $\mathbf{CO_i}$.

Lemma 1**:** Suppose $\mathbf{CO_i}$ at interval $I_{i,x}$ sends a message *M* to process P_k of **IN,** and $\mathbf{CO_j}$ receives a message *N* at interval $I_{j,y}$ from P_l of **IN**. If $C_{i,x}\rightarrow^{*}C_{j,y}$, then *N* must be sent after *M* is received**.**

Proof: Assume that *N* is sent before the receipt of *M*, we prove it by contradiction. Given $C_{i,x}\rightarrow^{*}C_{j,y}$, there exists $C_{k,z}$: If $k=l$, $C_{i,x}\rightarrow C_{k,z}$ (by *M*) and $C_{k,z}\rightarrow C_{j,y}$ (by *N*). But by rule 1, $C_{k,z}\rightarrow C_{k,z+1}$ and $C_{k,z}\rightarrow C_{j,y}$ (i.e., a new checkpoint set is taken upon the sending of *N*). Since P_k receives *M* after sending *N*, $C_{i,x}\rightarrow C_{k,z+1}$. Therefore, $C_{j,y}$ cannot be reached from $C_{i,x}$; Otherwise, if $k\neq l$, $C_{i,x}\rightarrow C_{k,z}$ (by *M*) and $C_{k,z}\rightarrow^{*}C_{j,y}$ (by *N*). But by rule 2, $C_{k,z}\rightarrow C_{k,z+1}$ and $C_{i,x}\rightarrow C_{k,z+1}$ (a new checkpoint is taken before receiving *M*), which again leads to a contradiction with $C_{i,x}\rightarrow^{*}C_{j,y}$. Thus *N* must be sent after *M* is received if $C_{i,x}\rightarrow^{*}C_{j,y}$. □

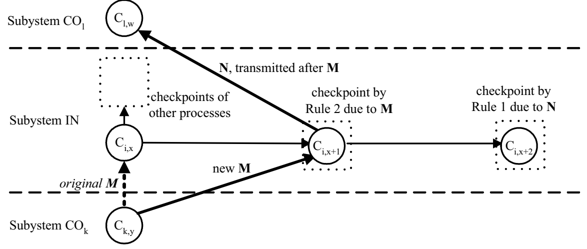
Theorem 1**:** If the unrelated messages are generated with a probability evenly distributed among the entire hybrid system, the proposed checkpointing algorithm can eliminate half of the unnecessary rollbacks between **CO**s.

Proof: By lemma 1, unnecessary rollback due to transitive dependency between two checkpoints is eliminated if *N* is sent before *M*. Since the unrelated message generation probability is assumed to be even across the entire hybrid system, *N* is equal likely to be sent before or after *M* if they are unrelated. Thus the proof for theorem 1 is completed. □

Theorem 2**:** If no further assumption is made on additional behavior for the subsystems and messages involved, the proposed checkpointing algorithm eliminates all the unnecessary rollbacks for **CO**s that are preventable.

Proof: If *N* is sent after *M*, it is not possible to tell whether *N* is related to *M*. Since it is not possible to reschedule the sending of *N* so that it is sent before *M* if they are unrelated but still do not require the assumption of additional behavior for the subsystems and messages involved, proof for theorem 2 is completed. □

The main weakness of the algorithm is that occasionally it may not be able to prevent one type of cross-subsystem rollbacks. Figure 5 illustrates such a situation. Suppose P_i of **IN** receives an application message *M* from $\mathbf{CO_k}$ after sending any application message during checkpointing interval $I_{i,x}$ to another process of **IN**, then P_i takes a new checkpoint $C_{i,x+1}$ by applying Rule 2. During interval $I_{i,x+1}$, if P_i sends a message *N* to $\mathbf{CO_l}$ (surely Rule 1 is applied), then there exists $\mathbf{CO_k}\rightarrow^{*}\mathbf{CO_l}$. Thus, if $\mathbf{CO_k}$ crashes subsequently, rollback to $C_{k,y}$ leads to rollback to $C_{l,w}$ in $\mathbf{CO_l}$.



Subsystem CO_l
N, transmitted after M
checkpoints of other processes
checkpoint by Rule 2 due to M
checkpoint by Rule 1 due to N
Subystem IN
new M
original M
Subsystem CO_k

Figure 5. A situation leading to cross-CO rollback

A careful study of this case indicates that it is actually the problem of integrating two coordinated checkpointing subsystems directly together. One of the solutions is to take $C_{k,y+1}$ immediately after the transmission of *M*, which implies that we don't have to perform actions proposed by Rule 2 (i.e., we don't need to take $C_{i,x+1}$ but rather have *M* transmitted to $C_{i,x}$) as it becomes unnecessary. However, in most cases, taking coordinated checkpointing is a much more time-consuming and efficiency-hurting action than taking an independent checkpoint of a single process. Thus actions proposed by Rule 2 are generally more preferable.

3.2. Recovery and Garbage Collection

When failures occur in the combined hybrid system, the recovery process can then be modeled by seeking *MAG* (Minimal Atomic Graph) [6] from the generalized checkpoint graph G, where *MAG* is the minimum subgraph induced by a set of nodes such that there are no outgoing arcs from *MAG* to other subgraphs of G. For the example shown in Figure 2, assume now P_0 fails and rollbacks to $C_{0,2}$. By finding out the MAG for $C_{0,2}$, we first obtain the *Rollback Set, R=MAG*($C_{0,2}$). We then obtain the *Final Rollback Set F* from R by: $F(C_{0,2}) = \{C_{x,y}: (C_{x,y} \in R) \wedge (\forall C_{i,j} \in R, i=x \Rightarrow y \leq j)\}$. In this example, $F(C_{0,2}) = \{C_{0,2}, C_{1,1}, C_{2,1}, C_{3,2}\}$. In fact, this approach is equivalent to the rollback-dependency graph [4] and checkpoint graph [26] approaches when being used to find the final rollback set.

For a checkpointing algorithm which requires each process to keep multiple checkpoints, garbage collection algorithms are required to relieve useless checkpoints. We observe that there is no need for garbage collection within each **CO** as any local checkpoint before the current consistent global checkpoint can be reclaimed; On the other hand, **IN** can adapt the optimal algorithm proposed in [25] to handle two exceptions corresponding to Rule 1 and Rule 2. When Rule 1 occurs, a new coordinated checkpoint set is taken, so all local checkpoints earlier than the new one can be reclaimed. For Rule 2, the incoming arc (message M) from any **CO** would not affect the garbage collection algorithm within **IN**. However, in order to keep track of the unavoidable cross-**CO** rollbacks discussed earlier, it is insufficient to keep only the current checkpoint sets of **CO**s, and a garbage collection algorithm is needed to decide which checkpoint sets of **CO**s must be retained. Our solution is to record, for each checkpoint set not yet being discarded, the identities of the checkpoint sets on which it depends. Whenever we transmit a message from **IN** to any **CO**, we update the dependency relationship correspondingly. Whenever any **CO** takes a new checkpoint, we mark the previous checkpoint sets as inactive if they no longer depend on any current checkpoint set.

The algorithm is shown in Algorithm 2. It is driven by the events of message exchanging between any **CO** and **IN** and checkpoint taking of any **CO**, together with the algorithm initialization. In Algorithm 2, $DCSet = \{((i,j),(x,y)) : C_{i,j} \rightarrow P_{x,y}\}$, which records the message transmission from **CO**s to **IN**, and $ASet_{i,j} = \{(x,y) : C_{x,y} \rightarrow^{*} C_{i,j} \wedge C_{x,y}$ is the current checkpoint for $\mathbf{CO_x}\}$, which is used to record the identities of the current coordinated checkpoint sets on which $C_{i,j}$ transitively depends. Besides, *ActiveSyncCkptSet* records the identities of the coordinated checkpoint sets that are still active (i.e., not reclaimed). The algorithm works in a straightforward way: When $\mathbf{CO_i}$ send a message to **IN**, a record is appended to *DCSet*. On the contrary, if the message is send from **IN** to $\mathbf{CO_i}$, then the cross-**CO** dependency tracking is triggered, and the involved *ASet*'s, if any, are updated. When $\mathbf{CO_i}$ takes a new synchronous checkpoint set $C_{i,j}$, each *ASet* associated with the coordinated checkpoint sets that depend on the previous checkpoint set $C_{i,j-1}$ is updated. If any *ASet* becomes empty, then its corresponding checkpoint set can be discarded. The correctness of Algorithm 2 is guaranteed by the following theorem.

Theorem 3: Algorithm 2 is correct in the sense that it can collect all useless coordinated checkpoint sets.

Proof: The proof is omitted due to space limitation. □

Algorithm 2 The algorithm for garbage collection

```
Event 1: initialization
1: $DCSet = \emptyset$;
2: $\forall i, ASet_{i,0} = \{(i,0)\}$;
3: $ActiveSyncCkptSet = \{(i,0)$ : for all $\mathbf{CO_i}$ and $P_i$ in $\mathbf{IN}\}$;
Event 2: When $\mathbf{CO_i}$ sends a message during $I_{i,j}$ to $P_\alpha$ in IN which
receives it during $I_{\alpha,\beta}$
1: $DCSet = DCSet \cup \{((i,j), (\alpha,\beta))\}$;
Event 3: When $P_\alpha$ in IN sends a message during $I_{\alpha,\beta}$ to $\mathbf{CO_i}$ which
receives it during $I_{i,j}$
1: $PossibleAsyncStartSet=\{(x,y):\exists k,l\ ((k,l),(x,y)) \in DCSet\}$;
2: for all $(x,y) \in PossibleAsyncStartSet$ {
3:    if $C_{x,y} \rightarrow^{*} C_{\alpha,\beta} \vee (x,y)==(\alpha,\beta)$ {
4:       $PossibleSyncStartSet=\{(\chi,\delta):((\chi,\delta),(x,y)) \in DCSet\}$;
5:       for all $(a,b) \in ActiveSyncCkptSet$ {
6:          if $(i,j) \in ASet_{a,b}$ {
7:             for all $(k,l) \in PossibleSyncStartSet$ {
8:                $ASet_{a,b} = ASet_{a,b} \cup ASet_{k,l}$;  } } } } }
Event 4: When $\mathbf{CO_i}$ takes a new coordinated checkpoint $C_{i,j}$
1: for all $(x,y) \in ActiveSyncCkptSet$ {
2:    $ASet_{x,y} = ASet_{x,y} - \{(i,j-1)\}$;
3:    if $ASet_{x,y} == \emptyset$ {
4:       Discard $C_{x,y}$;
5:       $ActiveSyncCkptSet = ActiveSyncCkptSet - \{(x,y)\}$;
6:       $DCSet=DCSet-\{((a,b),(c,d)):(((a,b), (c,d)) \in DCSet) \wedge$
         $((a,b) = (x,y))\}$;  } }
7: $ASet_{i,j} = \{(i,j)\}$;
8: $ActiveSyncCkptSet = ActiveSyncCkptSet \cup \{(i,j)\}$;
```

4. Conclusion

In this paper, a new algorithm is proposed to address the need of applying different checkpointing schemes to different subsystems inside a single target system. The proposed algorithm has several key advantages: (1) easy to implement; (2) only subsystems employing independent checkpointing schemes have to be modified. Almost no change is required for subsystems with coordinated checkpointing schemes; (3) relatively low extra workload for the coordinated checkpointing subsystems; and (4) can be applied at any time without having to shutdown the coordinated checkpointing subsystems.

If we consider the probability of generation of every inter-subsystem message to be equal, then the proposed algorithm reduces half of the unnecessary rollbacks. If we do not assume additional behavior for the subsystems and messages involved, then the proposed checkpointing algorithm eliminates all the unnecessary rollback for coordinated checkpointing subsystems that are preventable. Due to unavoidable cross-**CO** rollbacks, coordinated checkpointing subsystems cannot freely discard their old checkpoint sets. Therefore, a garbage collection algorithm is also presented. It is a passive algorithm and can make maximum usage of storage in the sense that it can determine immediately which checkpoint sets can be discarded after taking a new checkpoint set.

There are primarily two ways to improve the algorithm. Due to the fact that the passive garbage collection algorithm requires to execute from the beginning of system execution in order to successfully keep track of all the checkpoint sets that cannot be discarded, it prevents the algorithm from integrating at any time. Thus an active dependency tracking algorithm could alternatively be developed so as to allow flexible employment. Should such an active algorithm be developed, we can run it to get the active checkpoint sets, which enables to install the passive algorithm without the need to stop the operation of coordinated checkpointing subsystems. The best way, however, is to improve the algorithm so that coordinated checkpointing subsystems are always guaranteed to rollback to the most recent local consistent checkpoint set, which is the only instance of checkpoint sets required to be maintained in stable storage. Such a protocol can be developed with the help of message logging. We are currently working it in progress.

Acknowledgement

This work is partially supported by Hong Kong Polytechnic University under the research grant B-Q567 and National Natural Science Foundation of China under the research grant No. 90104005.

Reference

[1] T. Anderson, P.A. Lee, and S.K. Shrivastava, "System fault tolerance", in *Computing System Reliability*, T. Anderson, and B. Randell, Eds., Cambridge, MA: Cambridge University Press, 1979, pp. 153-210.

[2] G. Barigazzi, and L. Strigini, "Application-transparent setting of recovery points", *IEEE Proc. 3rd Int'l Conf. Distributed Computing Systems*, 1983, pp. 48-54.

[3] E. Best, and B. Ranbdel, "A formal model of atomicity in asynchronous systems", *Acta Informatica*, Vol. 16, 1981, pp. 93-124.

[4] B. Bhargava, and L. Lilien, "Independent checkpointing and concurrent rollback recovery for distributed systems - An Optimistic Approach", *IEEE Proc. 7th Symp. Reliability in Distributed Systems*, Oct 1988, pp. 3-12.

[5] D. Briatico, A. Giuffoletti, and L. Simoncini, "A distributed domino-effect free recovery algorithm", *IEEE Proc. 4th Symp. Reliability in Distributed Software and Database Systems*, Oct 1984, pp. 207-215.

[6] J. Cao, and K.C. Wang, "An abstract model of rollback recovery control in distributed systems", *ACM Oper. Sys. Review*, Oct 1992, pp.62-76.

[7] G. Cao, and M. Singhal, "On coordinated checkpointing in distributed systems", *IEEE Trans. Parallel and Distributed Systems*, Vol. 9, No. 12, Dec 1998, pp. 1213-1225.

[8] H. Casanova, "Distributed computing research issues in Grid computing", *ACM SIGACT News*, Vol. 33, No.3, Sep 2002, pp. 50-70.

[9] K.M. Chandy, and L. Lamport, "Distributed snapshots: determining global state of distributed systems", *ACM Trans. Computer Systems*, Vol. 3, No. 1, 1985, pp. 63-75.

[10]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, No. 3, Sep 2002, pp. 375-408.

[11]I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: enabling scalable virtual organizations", *Int'l J. High Performance Computing Applications*, Vol. 15, No. 3, 2001.

[12]D.B. Johnson, and W. Zwaenepoel, "Recovery in distributed systems using optimistic message logging and checkpointng", *J. Algorithms*, Vol. 11, 1990, pp. 462-491.

[13]R. Koo, and S. Toueg, "Checkpointing and rollback recovery for distributed systems", *IEEE Trans. Software Eng*., Vol. SE-13, No.1, Jan 1987.

[14]T.H. Lai, and T.H. Yang, "On distributed snapshots", *Information Processing Letters*, May 1987, pp. 153-158.

[15]P.J. Leu, and B. Bhargava, "Concurrent robust checkpointing and recovery in distributed systems", *IEEE 4th Conf. Data Eng.*, 1988, pp. 154-163.

[16]D. Manivannan, and M. Singhal, "Quasi-synchronous checkpointing: Models, characterization, and classification", *IEEE Trans. Parallel and Distributed Systems*, Vol. 10, No. 7, Jul 1999, pp. 703-713.

[17]J.A. Mcdermid, "Checkpointing and error recovery in distributed systems", *IEEE Proc. 2nd Intl. Conf. Distributed Computing Systems*, 1981, pp. 271-282.

[18]P.M. Merlin, and B. Randell, "State restoration in distributed systems", *IEEE Digest of Papers FTCS-8*, 1978, pp. 128-134.

[19]R.H.B. Netzer, and J. Xu, "Necessary and sufficient conditions for consistent global snapshots", *IEEE Trans. Parallel and Distributed Systems*, Vol. 6, No. 2, Feb 1995, pp. 65-169.

[20]M.L. Powell, and D.L. Presotto, "Publishing: A reliable broadcast communication mechanism", *ACM Proc. 9th Symp. Operating Systems Principles*, Oct 1983, pp. 100-109.

[21]B. Randel, "System Structure for Software Fault Tolerance", *IEEE Trans. Software Eng.*, Vol. SE-1, No. 2, 1975, pp.220-232.

[22]L.M. Silva, and J.G. Silva, "Global checkpointing for distributed programs", *Proc. 11th Symp. Reliable Distributed Systems*, Oct 1992, pp. 155-162.

[23]R. Strom, and S. temini, "Optimistic Recovery in Distributed Systems", *ACM Trans. Computer Systems*, Aug 1985, pp. 204-226.

[24]K. Tsuruoka, A. Kaneko, and Y. Nishihara, "Dynamic recovery schemes for distributed processes", *Proc. IEEE 2nd Symp. Reliability in Distributed Software and Database Systems*, 1981, pp. 124-130.

[25]Y.M. Wang, P.Y. Chung, I.J. Lin, and W.K. Fuchs, "Checkpoint space reclamation for uncoordinated checkpointing in message-passing systems", *IEEE Trans. Parallel and Distributed Systems*, Vol. 6, No. 5, May 1995, pp. 546-554.

[26]Y.M. Wang, "Consistent global checkpoints that contain a given set of local checkpoints", *IEEE Trans. Computers*, Vol. 46, No. 4, Apr 1997, pp. 456-468.

[27]W.G. Wood, "A Decentralized Recovery Control Protocol", *IEEE Symp. Fault Tolerant Computing*, 1981.