A Communication-Induced Checkpointing Algorithm using Virtual Checkpoint on Distributed Systems

Kim Do-Hyung and Park Chang-Soon
Department of S/W Engineering
Electronics and Telecommunications Research Institute, KOREA

Abstract

Checkpointing is one of the fault-tolerant techniques to restore faults and to restart job fast. The algorithms for checkpointing on distributed systems have been under study for years. These algorithms can be classified into three classes: coordinated, uncoordinated and communication-induced algorithms. In this paper, we propose a new communication-induced checkpointing algorithm that has a minimum checkpointing counts equivalent to periodic checkpointing algorithm, and relatively short rollback distance at faulty situations. The proposed algorithm is compared with the previously proposed communication-induced checkpointing algorithms with simulation results. In the simulation, the proposed algorithm produces better performance than other algorithms in terms of task completion time in both of fault-free and faulty situations.

1. Introduction

Checkpointing is an effective approach to tolerating both hardware and software faults [1-20]. During normal execution, the state of each process is periodically saved on stable storage as a checkpoint. When a fault occurs, the process can roll back to a previous checkpoint by reloading the checkpointed state. In a message-passing system, rollback propagation can occur when the rollback of a message sender results in the rollback of the corresponding receiver. The system is then required to roll back to the latest available consistent set of checkpoints, which is also called the recovery line, to ensure correct recovery with a minimum amount of rollback. In worst case, cascading rollback propagation may result in the domino effect, which prevents recovery line progression [8].

The checkpointing algorithms that have been suggested can be divided into three classes: uncoordinated, coordinated, and communication-induced checkpointing [1-3]. In the first class, each process takes its checkpoints independently and keeps track of the dependencies among checkpoints resulting from message communications. When a failure occurs, the dependency information is used to determine the recovery line to which the system should roll back. Uncoordinated checkpointing allows maximum process autonomy and has low checkpointing overhead. However, this approach may suffer from domino effect, in which the processes roll back recursively while determining a consistent set of checkpoints. And, uncoordinated checkpointing requires multiple checkpoints to be stored at each process.

In the second class, a checkpoint initiator process forces other processes, during a normal execution, to take a local checkpoint by using control message. In coordinated checkpointing, the last local checkpoint of each process belongs to a recovery line. So, the storage requirement for the checkpoints is minimum because each process keeps only one checkpoint in the stable storage at any given time. But, in coordinated checkpointing schemes, process execution may have to be suspended during the checkpointing coordinating, resulting in performance degradation.

In the third class, the coordination is lazy fashion by piggybacking control information on message. Each process takes some local checkpoints, namely periodic checkpoints, at its own space, then the lazy coordination induces some additional local checkpoints, namely message checkpoints. Communication-induced checkpointing schemes associate each local checkpoint with a checkpoint sequence number and try to enforce consistency among local checkpoints with the same sequence number. So, communication-induced checkpointing schemes have the easiness and low overhead of uncoordinated checkpointing and the recovery time advantages of coordinated checkpointing.

In this paper, we propose a new communication-induced checkpointing scheme, called virtual checkpointing, which reduces the checkpoint overhead compared to previously suggested communication-induced checkpointing algorithms and which has a relatively short rollback distance in faulty situation. In the proposed scheme, we use checkpoint sequence number to

0-7695-0568-6/00 $10.00 © 2000 IEEE

take a message checkpoint. But, unlike the previous communication-induced checkpointing algorithms, only one message checkpoint can exist between two consecutive periodic checkpoints. So, our checkpointing algorithm has smaller number of checkpoints than other communication-induced checkpointing algorithms. And, like the other communication-induced checkpointing algorithms, the dependency among checkpoints is removed by message checkpoints. In result, our checkpointing algorithm has a relatively short rollback distance in faulty situation. The advantages of our algorithm are investigated by a simulation showing that our algorithm has a better performance numbers than the others in terms of task completion time in both of fault-free and faulty situations.

This paper is organized as follows: Section 2 presents the system model and terminology. Section 3 describes the proposed algorithm. Section 4 presents correctness of the proposed algorithm. Section 5 presents the simulation and result. Finally, in section 6, we summarize our results.

2. System Model and Terminology

2.1. System model

We consider a distributed computation consisted of n processes $\{P_1, P_2, \ldots, P_n\}$ which interact with one another by message passing. Each pair of processes are connected by a two-way reliable channel whose transmission delay is unpredictable but finite. Processes are assumed to run on fail-stop processors and do not share a global memory or a global physical clock. In order to allow general non-deterministic execution, we do not assume the piecewise deterministic model. This implies that whenever the sender of a message m rolls back and unsends m, the receiver which has already processed m must also rollback to undo the effect of m because the potential non-determinism preceding the sending of m may prevent the same message from being resent during execution. We assume that each process use synchronous message logging to save received messages [19] and assume that checkpoints saved on stable storage are shielded from faults.

2.2. Terminology

During normal execution, each process saves its local checkpoints onto stable storage or memory. In this paper, a checkpoint on stable storage is called a stable checkpoint and a checkpoint on memory is called a virtual checkpoint. Let $CP_{i,x}$ denote the xth stable checkpoint($x \geq 0$) of process $P_i(0 \leq i \leq n-1)$ and $VCP_{i,x}$ denote the xth virtual checkpoint(x 0) of process $P_i(0 \leq i \leq n-1)$. When the content of $VCP_{i,y}$ is written to $CP_{i,x}$, $CP_{i,x}$ is a transition checkpoint from $VCP_{i,y}$. Let $I_{i,x}$ denote the xth checkpoint interval of process P_i between consecutive checkpoints $CP_{i,x}$ and $CP_{i,x+1}$. Current checkpoint interval is the interval between latest checkpoint and current execution point.

3. Virtual Checkpointing Algorithm

In the virtual checkpointing algorithm, each process takes periodic checkpoints, which are saved onto stable storage, at each CKP(Checkpoint) time. And, like the other previous communication-induced checkpointing algorithms, the dependencies among checkpoints are removed by message checkpoints, which are saved in memory. But, unlike the other communication-induced checkpointing algorithms, only one message checkpoint can exist between two consecutive periodic checkpoints. So, our checkpointing algorithm has smaller number of checkpoints and has relatively short rollback distance in faulty situations than the other algorithms. Virtual checkpointing algorithm is shown in Figure 1.

```
/* for process P_i , 0 ≤ i ≤ n-1 */
While (P_i is not ended)
{
  if (current_time = CKP) /* P_i has to checkpoint */
  {
    if (virtual checkpoint flag is True in current interval)
      Take a stable checkpoint with virtual checkpointed
contents;
    else
      Take a stable checkpoint with current process state;

   Increase checkpoint number by 1;
   CKP = current_time + CKP;
   Set virtual checkpoint flag to False;
  }
  ...
  /* if P_i has to process a received message */
  If (senders checkpoint number > current checkpoint
number)
  {
   if (virtual checkpoint flag is False in current interval )
    {
     Take a virtual checkpoint with current process state;
     Set current checkpoint number to senders;
     Set virtual checkpoint flag is True;
    } else
     Set current checkpoint number to senders;
  }
  Process a received message;
}
```

Figure 1. Virtual checkpointing algorithm

With the virtual checkpointing algorithm, each process takes a periodic checkpoint at each CKP time. At this point, before taking a checkpoint, each process investigates if there is a virtual checkpoint in current interval. If one exists, then the content of virtual checkpoint, which is stored in memory, is written to a stable storage. Otherwise, the current state information of the process is written to a stable storage. After checkpointing, a checkpoint sequence number is increased by 1 and CKP time is updated.

During normal execution, each process takes a message checkpoint before processing a received message. Each process, before processing message, compares its current checkpoint number with the checkpoint number of message sender that is tagged on each message. If the checkpoint number of message sender is bigger than the current checkpoint number, then each process investigate if there is a virtual checkpoint in current checkpoint interval. If a virtual checkpoint is exists, then set the current checkpoint number to the message sender's checkpoint number and process the received message. Otherwise, the current state information of process is written to memory (virtual checkpoint) and set the current checkpoint number to the message sender's checkpoint number. After taking virtual checkpoint, each process processes the received message.

The virtual checkpointing algorithm always has a minimum stable checkpoint number regardless of the difference of process execution time or the amount of message traffic. Of course, virtual checkpointing algorithm demands taking a large number of virtual checkpoints. And, in worst case, virtual checkpointing algorithm may take virtual checkpoints equal to stable checkpoints. However, the overhead of virtual checkpoint is much smaller than the one by stable checkpoint. And, by removing the dependency among checkpoints with virtual checkpoint, the proposed algorithm has relatively short rollback distance at faulty situation. Checkpointing count is directly related to task completion time in a fault-free situation and short rollback distance is to task completion time in a faulty situation. Therefore, virtual checkpointing algorithm produces better performances than previous communication-induced checkpointing algorithm in terms of task completion time in fault free and faulty situations.

4. Correctness Proof

We want to prove that at any time virtual checkpointing algorithm can determine the recovery line.

Observation 4.1 If $VCP_{j,m}$ is induced by message sent from $CP_{i,k}$, then m = k.

Observation 4.2 If $CP_{i,n}$ is a transition checkpoint from $VCP_{i,m}$, then the content of two checkpoints is equal and n > m.

Observation 4.3 If $VCP_{j,m}$ is induced by message sent from $CP_{i,k}$ and $CP_{j,n}$ is a transition checkpoint from $VCP_{j,m}$, then the contents of $CP_{j,n}$ and $CP_{i,k}$ are consistent.

Observation 4.4 For any two checkpoints $CP_{i,m}$ and $CP_{j,n}$, if there are no messages among them, then the contents of $CP_{i,m}$ and $CP_{j,n}$ are consistent.

Lemma 4.5 For any two checkpoints $CP_{i,l}$ and $CP_{j,n}$, if l = n, then the contents of $CP_{i,l}$ and $CP_{j,n}$ are consistent.

Proof. Assume that two checkpoint $CP_{i,l}$ and $CP_{j,n}$(l = n) do not constructed a recovery line. Then, an orphan message is exist from $CP_{i,l}$ to $CP_{j,n}$, or vice versa. But, this situation is impossible. According to virtual checkpointing algorithm, process P_j takes a virtual checkpoint $VCP_{j,m}$(m=l) before processing message and set checkpoint number to the checkpoint number of P_i. And, when P_j writes the content of $VCP_{j,m}$ to stable storage at periodic checkpoint time, checkpoint number is increased by 1(m < n). Then, checkpoint number n is larger than l. But, this contradicts to previous assumption. Therefor any pair of checkpoints that has same checkpoint number in S constructs a recovery line.

Lemma 4.6 If there is a message, sent from $CP_{i,l}$, between checkpoints $CP_{j,k}$ (k < l) and $CP_{j,n}$, then $CP_{i,l}$ and $CP_{j,n}$ are consistent and satisfy k < l < n.

Proof. By virtual checkpointing algorithm, $VCP_{j,m}$(k < m = l) occurs between $CP_{j,k}$ and $CP_{j,n}$. By Observation 4.1, 4.2, 4.3, 4.4, two checkpoints $CP_{i,l}$ and $CP_{j,n}$ are consistent and satisfy k < l < n.

Theorem 4.7 If any pair of checkpoint $CP_{i,k}$ and $CP_{j,m}$ of global checkpoint set S have same checkpoint number or if $CP_{j,m}$ is a transition checkpoint from $VCP_{j,k}$, induced by $CP_{i,k}$ and when $CP_{j,l}$ is a previous stable checkpoint before $CP_{j,m}$ and l < k < m is satisfied, then S is a recovery line.

Proof. The proof follows from Lemma 4.5 and Lemma 4.6.

5. A Performance Study

5.1. The Simulation Model

The simulation compares periodic, lazy [1], MS [2], and virtual checkpointing algorithms. In periodic checkpointing algorithm, each process takes a stable checkpoint at CKP time. So, periodic checkpointing algorithm always has the minimum checkpoint count. In simulation, we measured the task completion time in a fault-free situation and in one fault situation. Task completion time is an appropriate factor to compare

checkpointing algorithms because it shows how fast job is finished regardless of faults. And, in simulation, we use the recovery algorithm that is described in [3]. In this paper, we omit detailed recovery algorithm.

In simulation, there are four process groups that take checkpoints and process messages according to each checkpointing algorithm. Each process group consists of 10 processes and executes the same job. In simulation, we transfer same messages to each group. In other words, if a process included in the first process group sends a message to other process, which is in same group, then same thing also occurs in other groups. So, four process groups have the same message traffic. In simulation, messages are randomly generated using message generation ratio whose range varies from 0 to 0.1.

In simulation, to differentiate the process execution time, we use the method that is used in [14]. In that method, the checkpoint time of each process is calculated like CKP – (0 ~ 9) * , where is a checkpoint offset. This method can differentiate the process execution time even if the difference of process execution time is similar during execution. In simulation, we randomly set to 2, 4, the delay of stable checkpoint to 10, the delay of virtual checkpoint to 0.1, CKP to 300, and total task clock to 6000. In simulation, is randomly set for the difference of process execution time to be formed in the 12 % of CKP time [14]. And, virtual checkpoint time is randomly set to 1 % of stable checkpoint time and CKP time is also randomly set to 5 % of total task clock. Finally, at each message generation ratio, task is repeatedly executed to 10 times and task completion time is averaged. And, at each execution time, the message transfer situation is different from the previous one.

5.2. Results of the Experiments

5.2.1. The task completion time of fault free situation. Figure 2 shows the task completion time of each checkpointing algorithm in a fault free situation.

Simulation shows that periodic and virtual checkpointing algorithms have smaller task completion time at fault-free situations. Lazy checkpointing algorithm has bigger task completion time than others. In fault-free situation, task completion time is directly reflected to checkpoint count. So, periodic checkpointing algorithm always has minimum task completion time in fault-free situations. Virtual checkpointing algorithm has the same number of stable checkpoints and a number of virtual checkpoints. But, because the overheads of virtual checkpoint are much less than the one from stable checkpointing algorithm, the virtual checkpointing algorithm has smaller task completion time compatible to that of periodic checkpointing algorithm. MS checkpointing algorithm has relatively small number of checkpoints because of the checkpoint skip rule. But, simulation shows that checkpoint count is increased as both of the difference of process execution time and the amount of message are increased. Lazy checkpointing algorithm has bigger task completion time because of large number of message checkpoints. In lazy checkpointing algorithm, checkpoint count is increased as the difference of process execution time and the amounts of message are increased.

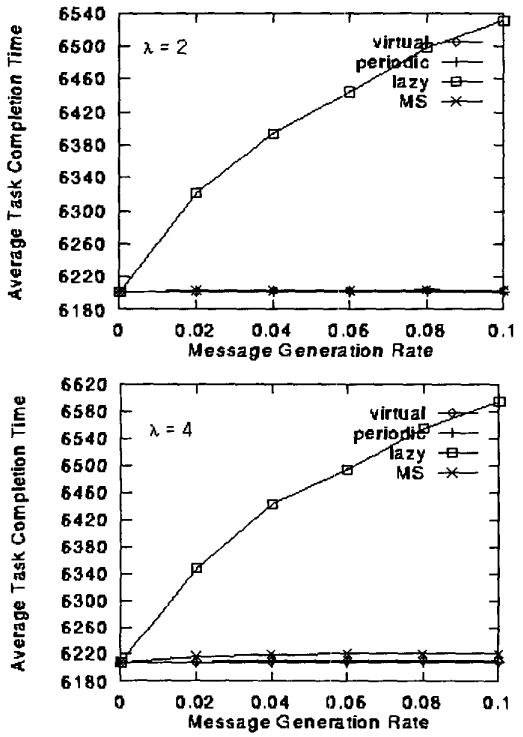

Figure 2. The task completion time of fault free situation

This simulation shows that periodic and virtual checkpointing algorithms have better performance numbers in terms of task completion time in fault free situations

5.2.2. The task completion time of one fault situation. Figure 3 shows the task completion time of each checkpointing algorithms in one fault situation.

Simulation shows that the virtual checkpointing algorithm has smaller task completion time, but periodic checkpointing algorithm has bigger task completion time. Periodic checkpointing algorithm has minimum number of

checkpoints. But, when a fault occurs, the rollback distance is much bigger than other algorithms, so task completion time is worst. Lazy checkpointing algorithm has smaller rollback distances than other algorithms, but because of considerable number of checkpoints, task completion time is bigger than the one of virtual checkpointing algorithm. MS checkpointing algorithm has small number of checkpoints. But, because of the checkpoint skip rule, when a fault occurs, the rollback distance number is bigger than virtual checkpointing algorithm. And, as the difference of process execution time and message traffic is got increased, task completion time also gets increased.

The proposed checkpointing algorithm takes a virtual checkpoint before processing a message and sets virtual checkpoint number to message senders to reduce the dependencies among checkpoints. As a result, virtual checkpoint performs the role of reducing the rollback distance, which is induced by dependencies among checkpoints. So, virtual checkpointing algorithm has better performance with minimum number of stable checkpoints equal to the periodic checkpointing algorithm.

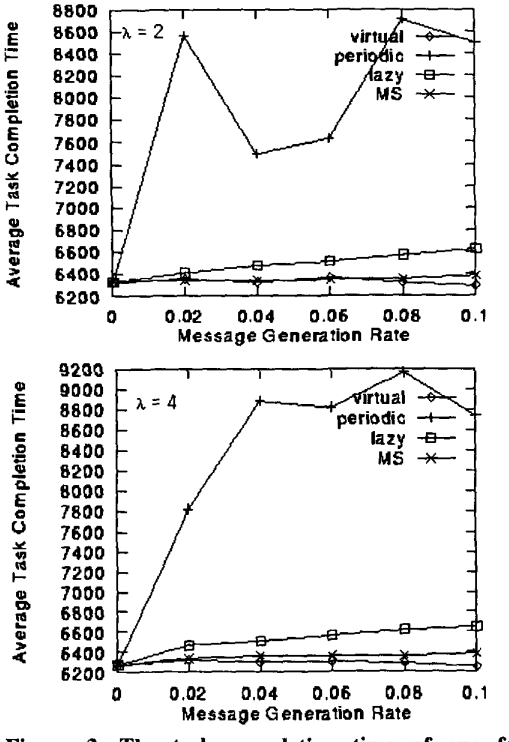

Figure 3. The task completion time of one fault situation

This simulation shows that the virtual checkpointing algorithms have better result than other algorithms in terms of task completion time in faulty situation.

6. Conclusion

In this paper, we have presented a new communication-induced checkpointing algorithm, called virtual checkpointing algorithm. Virtual checkpointing algorithm has small number of checkpoints, which is comparable to periodic checkpointing algorithm regardless of the difference of process execution time and the amount of messages. So, virtual checkpointing algorithm has the short task completion time in fault free situations. And, by removing the dependencies among checkpoints from virtual checkpoint, which is induced by message passing, the proposed algorithm achieves relatively short rollback distances in fault situations. Simulation shows that the virtual checkpointing algorithm produces better performance than other algorithms in terms of task completion time in both of fault free situations and faulty situations.

7. References

[1] Y. -M. Wang and W. K. Fuchs, "Lazy Checkpoint Coordination for Bounding Rollback Propagation", *Technical Report CRHC-92-27*, Center for Reliable and High-Performance Computing, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Nov. 1992.

[2] D. Manivanan and M. Singhal, "A Low-Overhead Recovery Technique Using Quasi Synchronous Checkpointing", *Proc. IEEE Int'l Conf. Distributed Computing Systems*, 1996, pp. 100-107.

[3] J. Xu and R. H. B. Netzer, "Adaptive Independent Checkpointing for Reducing Rollback Propagation", *Proc. of the Fifth IEEE Symp. on Parallel and Distributed Processing*, Dec. 1993, pp. 754 - 761.

[4] J. Xu and R. H. B. Netzer, "Necessary and Sufficient Conditions for Consistent Global Snapshots", *IEEE Trans. on Parallel And Distributed Systems*, Vol. 6, No.2, Feb. 1995.

[5] Y.-M. Wang, "Consistent Global Checkpoints that Contain a Given Set of Local Checkpoints", *IEEE Trans. on Computers*, Vol. 46, No. 4, April 1997, pp. 456 - 468.

[6] R. E. Strom and S. Yemini, "Optimistic Recovery in Distributed Systems", *ACM Trans. on Computer Systems*, Vol. 3, No. 3, Aug. 1985, pp. 204 - 226.

[7] T. T-Y Juang and S. Venkatesan, "Crash Recovery with Little Overhead", *The 11th Int'l Conf. on Distributed Computing Systems*, May 1991, pp. 454 - 461.

[8] 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. on Parallel And Distributed Systems*, Vol. 6, No.5, May 1995.

[9] E.N. Elnozahy, D.B. Johnson, and W. Zwaenpoel, "The performance of consistent Checkpointing", *Proc. of the 11th Symp. on Reliable Distributed Systems*, Oct. 1992, pp. 39 -47.

[10] K. Venkatesh, T. Radhakrishnan and H. F. LI, "Optimal Checkpointing And Local Recording For Domino-Free Rollback Recovery", *Information Processing Letter*, Vol. 25, No. 5, July 1987, pp. 295 - 304.

[11] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", *ACM Trans. on Computer Systems*, Vol. 3, No. 1, Feb. 1985, pp. 63 - 75.

[12] R. D. Schichting and F. B. Schneider, "Fail-stop processors: An approach to designing fault-tolerant computing systems", *ACM Trans. on Computer Systems*, Vol. 1, Aug. 1983, pp. 222 - 238.

[13]R. Koo and S. Toueg, "Checkpointing and Rollback-Recovery for Distributed Systems", *IEEE Trans. on Soft. Eng.* Vol. SE-13, No.1, Jan., 1987, pp.23-31.

[14] Y.-M. Wang and W. K. Fuchs, "Scheduling Message Processing for Reducing Rollback Propagation", *FTCS-22*, 1992, pp.204-211.

[15] H.-C. Nam, J. Kim, S. Hong, and S. Lee, "Probabilities Checkpointing", *FTCS-27*, 1997, pp.48-57.

[16] D. B. Johnson and W. Zwaenpoel, "Recovery in Distributed Systems Using Optimistic Message Logging and Checkpointing", *J. of Algorithms*, Vol. 11, 1990, pp.462-491.

[17] H. V. Leong and D. Agrawal, "Using Message Semantics to Reduce Rollback in Optimistic Message Logging Recovery Schemes", *Int'l Conf. on Distributed Computing Systems*, 1994, pp.227-234.

[18] T. H. Lin and K.G. Shin, "Damage Assessment for Optimal Rollback Recovery", *IEEE Trans. on Computers,* Vol.47, No.5, May, 1998, pp.603-613.

[19] L. Alvisi and K. Marzullo, "Message Logging Pessimistic, Optimistic, Causal, and Optimal", *IEEE Trans. on Soft. Eng.*, Vol.24, No.2, Feb., 1998, pp.149-159.

[20] B. W. Johnson, *Design and Analysis of Fault Tolerant Digital Systems*, Addison-Wesley Publishing Company, 1989.