International Journal of Information Technology and Knowledge Management
January-June 2008, Volume 1, No. 1, pp. 25-33

MINIMUM PROCESS ERROR RECOVERY ALGORITHMS FOR MOBILE DISTRIBUTED SYSTEM USING GLOBAL CHECKPOINT

Surender Kumar, R. K. Chauhan & Parveen Kumar

Abstract

A checkpoint algorithm for mobile computing systems needs to handle many new issues like: mobility, low bandwidth of wireless channels, and lack of stable storage, disconnections, limited battery power and high failure rate of mobile hosts [MHs]. These issues make traditional checkpointing techniques unsuitable for checkpointing mobile computing systems. Minimum-process coordinated checkpointing is an attractive approach to introduce fault tolerance in mobile distributed systems transparently. The number of processes that take checkpoints is minimized to avoid awakening of MHs in doze mode of operation and to control the thrashing of MHs with checkpointing activity. It also saves limited battery life of MHs and low bandwidth of wireless channels. This approach is domino-free, requires at most two checkpoints of each process on stable storage; and in case of a fault, processes rollback to last consistent global state for recovery. But, it requires blocking of the underlying computation during checkpointing or taking some useless checkpoints. In this paper, we present a single phase non-blocking coordinated checkpointing algorithm suitable for mobile computing environments. It produces a consistent set of checkpoints, without the overhead of taking temporary checkpoints; the algorithm makes sure that only minimum number of processes are required to take checkpoints in any execution of the checkpointing algorithm; it uses very few control messages and the participating processes are interrupted less number of times. Performance analysis shows that our proposed approach outperforms some existing important related works.

Keyword: Coordinated Checkpointing, Non-blocking approach, Mobile Computing Systems.

1. Introduction

Checkpointing/rollback-recovery strategy has been an attractive approach for providing fault-tolerance to distributed applications. A checkpoint is a snapshot of the local state of a process, saved on local nonvolatile storage to survive process failures. A global checkpoint of an *n*-process distributed system consists of *n* checkpoints (local) such that each of these *n* checkpoints corresponds uniquely to one of the *n* processes. A global checkpoint *M* is defined as a consistent global checkpoint or state (CGS) if no message is sent after a checkpoint of *M* and received before another checkpoint of *M* [1]. The checkpoints belonging to a consistent global checkpoint are called globally

consistent checkpoints (GCCs). Checkpointing algorithms are classified into two main categories: (a) coordinated and (b) uncoordinated. In uncoordinated check pointing approach each process takes its checkpoint independently without the knowledge of the other processes. While the checkpointing approach is simple, yet it may suffer from the domino effect during recovery. In case of a failure; after recovery a CGS is found from the existing checkpoints and the system restarts from there. In coordinated checkpointing approach, all processes synchronize through control messages before taking checkpoints.

2. RELATED WORKS AND PROBLEM FORMULATION

The research in the area of coordinated checkpointing concentrates mostly on non-blocking approaches. In [3] and [4] the authors have proposed non-blocking coordinated checkpointing algorithms that require only a minimum number of processes to take checkpoints at any instant of time. In [8], it is a non-blocking coordinated check pointing algorithm; however minimality is not guaranteed. Let us now briefly state the working principles of these approaches and also the number of control messages these algorithms generate. In the following discussion, by 'number of phases of an algorithm' we mean the number of times an initiator process interacts with the other processes during the execution of the algorithm.

In [3] the authors have introduced the concept of mutable checkpoints. They are neither temporary nor permanent checkpoints. The basic idea about the algorithm is as follows: in the first phase an initiator process first checks its own dependency vector from which it finds the processes which have sent at least one computational message to the initiator. It then sends check pointing request messages to all such dependent processes, i.e. the processes from each of which the initiator process has received at least one message. These dependent processes after taking tentative checkpoints in turn send checkpoint requests to processes that are dependent on them. This goes on until there is no further dependency found. Suppose that the minimum number of processes which need to take checkpoints is n_{min}. In the second phase processes which have received checkpoint requests send reply to the initiator. The number of such reply messages is n_{min}. In the third phase, the initiator process sends the commit message to the processes asking them to convert their tentative checkpoints to permanent ones. Here the initiator selects between the minimum cost of broadcasting the control message to n processes and the cost of sending the messages to n_{min} processes. The total number of control messages in [3] is $2*n$ min + min (n_{min}, n_{broad}) where n_{broad} is the number of control messages required to broadcast to all the processes in the system. This calculation is not exact as it does not consider more complicated situations. For example, if any process which receives a check pointing request and sends a computational message to another process after taking a checkpoint, the process receiving the computational

message takes a mutable checkpoint first and computes the message. This mutable checkpoint is later converted to a permanent checkpoint if it receives a check pointing request; otherwise it becomes a useless checkpoint.

3. PROBLEM FORMULATION

The objective of the present work is to design a check pointing algorithm that is suitable for mobile computing environment. Mobile computing environment demands efficient use of the limited wireless bandwidth and the limited resources of mobile machines, such as battery power, memory etc. Observe that taking a temporary or a tentative checkpoint and later converting it to a permanent one needs more control messages. As a result it affects the bandwidth utilization negatively as well as it results in large number of interrupts to the mobile hosts; thereby wasting the mobile hosts' limited battery power. Therefore in the present work we emphasize on eliminating the overhead of taking temporary (tentative) checkpoints. To summarize, we have proposed a non-blocking coordinated checkpointing algorithm in which processes take permanent checkpoints directly without taking temporary checkpoints and whenever a process is busy, the process takes a checkpoint after completing the current procedure. As will be shown later that the proposed algorithm in this paper requires much fewer control messages and hence, fewer number of interrupts to each participating process compared to the other coordinated checkpointing works [3], [4], and [8]. Besides as in [3] and [4], our proposed algorithm requires only minimum number of processes to take checkpoints. It makes our algorithm suitable for mobile computing environments.

This paper is organized as follows. In Section 3 we state the system model considered in this work. We also state the date structures needed and give an example that shows how our idea works. It also contains some relevant observations. In Section 4 we have stated the algorithm. Section 5 contains a detailed performance comparison with some noted related works. In Section 6, we have discussed the suitability of our proposed algorithm in the mobile computing environment. Finally Section 7 draws the conclusion.

4. SYSTEM MODEL AND DATA STRUCTURES

The distributed system has the following characteristics. Processes do not share memory and they communicate via messages sent through channels. Channels can lose messages. However, they are made virtually lossless and order of the messages is preserved by some end-to-end transmission protocol. When a process fails, all other processes are notified of the failure in finite time. We also assume that no further processor (process) failures occur during the execution of the algorithm. In fact, the algorithm may be restarted if there are further failures. Consider a set of n processes, $\{P_1, P_2, \ldots, P_n\}$ involved in the execution of a distributed algorithm. Each process P_i maintains a dependency vector DV_i of size n which is initially empty and an entry $DV_i[j]$ is set to 1

when P_i receives since its last checkpoint at least one message from P_j. It is reset to 0 again when process P_i takes a checkpoint. Each process P_i maintains a checkpoint sequence number csn_i. This csn_i actually represents the current check pointing interval of process P_i. The i^{th} checkpointing interval of a process denotes all the computation performed between its i^{th} and $(i + 1)^{th}$ checkpoint, including the i^{th} checkpoint but not the $(i + 1)^{th}$ checkpoint. The csn_i is initially set to 1 and is incremented when process P_i takes a checkpoint. In this approach we assume that only one process can initiate the check pointing algorithm. This process is known as the initiator process. We define that a process P_k is dependent on another process P_r, if process P_r since its last checkpoint has received at least one application message from process P_k. In our proposed algorithm we assume primary and secondary checkpoint request exchanges between the initiator process and the rest $n - 1$ processes. A primary checkpoint request is denoted by R_i $(i = csn_j)$ where i is the current checkpoint sequence number of process P_j that initiates the checkpointing algorithm. It is sent by the initiator process P_j to all its dependent processes asking them to take their respective checkpoints. A secondary checkpoint request denoted by R_{si} is sent from a process P_m to a process P_n which is dependent on P_m to take a checkpoint. R_{si} means to its receiver process that I is the current checkpoint sequence number of the sender process. The control message exchange is explained with an illustration shown in Fig. 1.

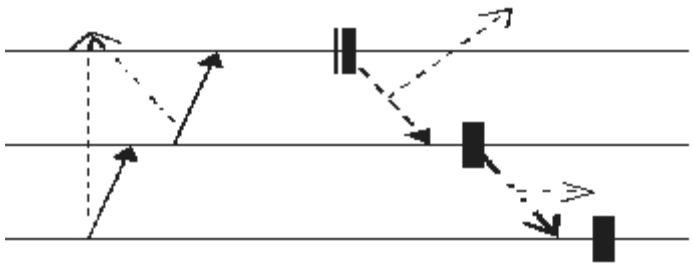

Figure 1: Control Message Exchanges in Our Approach

Consider a distributed system with three processes P_1, P_2, and P_3. We assume that P_1 initiates the checkpointing algorithm. To start with, P_1 takes a checkpoint and sends a primary checkpoint request to P_2, asking it to take a checkpoint as it is directly dependent on P_1. P_2 takes a checkpoint after it receives the primary checkpoint request. After taking its checkpoint P_2 sends a secondary checkpoint request to P_3 as P_3 is dependent on P_2, Process P_3 then takes its checkpoint. In this work, an application message is represented by $M_{i,x}$, which means that it is the x^{th} message sent by process P_i. The checkpoint $C_{i,j}$ represents the j^{th} checkpoint taken by P_i. We have assumed that the events of taking a checkpoint and sending a checkpoint request are done atomically. Also, each process P_i piggybacks its current checkpoint sequence number with only every first outgoing application message to another process after taking We now state

the situations in general when a process P_i needs to take a checkpoint. In our approach a process P_i takes a checkpoint if any of the following events occurs:

1. if P_i is the initiator
2. if it receives a primary checkpoint request from the initiator
3. the first time it receives a secondary checkpoint request and prior to that it has not received any primary checkpoint request or any piggybacked application message.
4. the first time it receives an application message piggybacked with the checkpoint sequence number, and prior to that it has not received any primary or secondary checkpoint request message.

5. SINGLE PHASE NON-BLOCKING COORDINATED CHECKPOINTING ALGORITHMS

As in any conventional coordinated check pointing scheme at any instant of time any one process can initiate the check pointing algorithm. The responsibility of the initiator process and all other processes are stated below.

Initiator process *Pi*

Step 1: take a checkpoint, check the dependency vector *DVi*[];

Step 2: when *DVi*[*k*] = = 1 for 1<= *k*<= *n*

Send a Primary request-*Rn* to process *Pk*;

/* checks the dependency vector and multicasts a checkpoint request */

Step 3: increment the checkpoint sequence number csni;

Step 4: continue normal computation;

if any secondary checkpoint request is received

discard it and continue normal execution;

Any Process *Pj j*! = *i* and 1 <= *j* <= *n*

if *Pj* receives a primary checkpoint request from *Pi*

take a checkpoint; /* if *Pj* is busy with other high priority job, it takes a checkpoint after the

job ends; otherwise it takes a checkpoint immediately */

if DVj[] = null;

increment csnj;

continue computation;

else

send secondary checkpoint request to each *Pk* such that *DVj*[*k*] = 1;

increment *csnj*;

continue computation;

else if *Pj* receives a secondary checkpoint request

if *Pj* has already participated in the checkpointing algorithm

/* csnj is greater than the received checkpoint sequence number*/

ignore the checkpoint request and continue computation;

else

take a checkpoint; /* if *Pj* is busy with other high priority job, it takes a checkpoint after

if *DVj* [] = null; the job ends; otherwise it takes a checkpoint immediately */

increment csnj;

continue computation;

else

send secondary checkpoint request to each *Pk* such that *DVj*[*k*] = 1;

increment *csnj*;

continue computation;

else if *Pj* receives a piggy backed application message

if Pj has already participated in the checkpointing algorithm

/* *csnj* is greater than the received checkpoint sequence number*/ process the message and continue computation;

else /* if *Pj* is busy with other high priority job, it takes a checkpoint take a checkpoint;

after the job ends; otherwise it takes a checkpoint immediately*/ if *DVj* [] = null;

increment *csnj*; process the message; continue computation;

else

send secondary checkpoint request to each *Pk* such that *DVj*[*k*] = 1; increment *csnj*;

process the message; continue computation;

Consider the pseudo code for any process P_j. Process P_j makes sure that all processes from which it has received messages also take checkpoints so that there are no orphan messages that it has received. In the second else if block of the pseudo code, process P_j first takes its checkpoint if needed, then processes the received piggybacked application message. Hence such a message cannot be an orphan. Hence the algorithm generates a consistent global state (CGS) of the system. ¦

6. PERFORMANCE

The main advantage of our algorithm over the algorithms [3], [4], and [8] is that the cost for determining a consistent state of the system is much less compared to the ones in [3], [4], and [8]. We have presented the comparison of performance of the above three algorithms with our algorithm in Table 1.

Table 1
Performance Comparison of the Checkpointing Algorithms

	Mutable [3]	*Non intrusive [4]*	*CCUML [8]*	*Our algorithm*
Cost (best case)	$2 \min * C_{air} + \min (n_{min} * C_{air}, n_{broad})$	$n * C_{air} + 2 * n_{min} * C_{air} + 2 * n_{broad} + 2 * n_{broad}$	$n_{min} * C_{air} + 2 * n_{broad}$	$n_{min} * C_{air}$
Useless checkpoints	present	nil	nil	nil
Temporary checkpoints	present	present	present	no
Non-Blocking	Yes	Yes	Yes	Yes
Number of checkpoints	n_{min}	n_{min}	$n + 1$	n_{min}

For ease of interpretation of the performance parameters we consider an $n + 1$ process distributed system. Let n_{min} represent the minimum number of processes that need to take a checkpoint, C_{air} be the cost of sending a message from one process to another, and n_{broad} be the cost of broadcasting a message to all processes in the system.

The cost to complete the checkpoint process using algorithm [3] is given as $2 * n_{min} * C_{air} + \min (n_{min} * C_{air}, n_{broad})$ in the best case. As mentioned earlier, in algorithm [3] first the initiator sends control messages to minimum number of processes that need to take a checkpoint each. The cost for this is $n_{min} * C_{air}$. When each process takes a tentative checkpoint it replies back to the initiator acknowledging the request to take a checkpoint. Hence a cost of $2 * n_{min} * C_{air}$ is needed. When the initiator receives the acknowledgement from all the processes, it informs them to convert their respective tentative checkpoints into permanent checkpoints which contributes further a cost of $n_{min} * C_{air}$. If the cost of broadcasting the message is less than sending the messages to n_{min} processes then the message can be broadcasted. In this way, the algorithm in [3] generates a consistent set of checkpoints. The total cost for such a generation is

$2 * n_{min} * C_{air} + \min (n_{min} * C_{air}, n_{broad})$ in the best case (secondary and tertiary dependencies are not considered).

In [4] the initiator broadcasts dependency vector request to all the n processes the cost of which is n_{broad}. The initiator receives the vectors from the n processes the cost of which is $n * C_{air}$. Initiator calculates the minimum dependency set from the dependency vectors and sends checkpoint request message to the minimum number of processes that need to take checkpoints, the cost of which is $n_{min} * C_{air}$. These processes reply to the initiator after taking temporary checkpoints, the cost of which is $n_{min} * C_{air}$. Finally the initiator broadcasts a commit message to all the processes, the cost of which is n_{broad}. In this way the algorithm in [4] generates a consistent set of checkpoints. The cost for such a generation is $n * C_{air} + 2 * n_{min} * C_{air} + 2 * n_{broad}$.

In [8] the initiator broadcasts the checkpoint request to all processes the cost of which is n_{broad}. The initiator receives replies from the n processes the cost of which is $n * C_{air}$. Finally the initiator broadcasts a commit message to all processes to convert their temporary checkpoints to permanent ones, the cost of which is which is n_{broad}. Hence the total cost in [8] is $n * C_{air} + 2 * n_{broad}$. Note that it does not guarantee that only minimum number of processes will take checkpoints.

CONCLUSION

In this paper, we have presented a single phase non-blocking coordinated checkpointing approach suitable for mobile computing environment. The main features of the algorithm are: (1) it is free from the avalanche effect and minimum number of processes take checkpoints; (2) it does not take any temporary, tentative, or mutable checkpoint unlike in some other important related works [3], [4], [8]. Absence of temporary, tentative, or mutable checkpoints (hence some possible useless checkpoints) means that much fewer number of control messages are needed. These advantages make the proposed algorithm more suitable for mobile computing environment than the algorithms mentioned above.

REFERENCES

[1] K. M. Chandy and L. Lamport, (February-1985), "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Transactions Computer Systems*, **3**(1), 63-75.

[2] G. Cao and M. Singhal, (December-1998), "On Coordinated Checkpointing in Distributed Systems," Parallel and Distributed Systems, *IEEE Transactions on Parallel and Distributed Systems*, **9**(12), 1213-1225.

[3] G. Cao and M. Singhal, (February-2001), "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, **12**(2), 157-172.

[4] P. Kumar, L. Kumar, R. K. Chauhan and V. K. Gupta, (January-2005), "A Non-intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems,"

ICPWC 2005, IEEE International Conference on Personal Wireless Communications, 491-495, New Delhi.

[5] E. N. Elnozahy, D. B. Johnson and W. Zwaenepoel, (October-1992), "The Performance of Consistent Checkpointing," Proceedings of 11th Symp. *On Reliable Distributed Systems*, 86-95, Houston.

[6] L. M. Silva and J. G. Silva, (October-1992), "Global Checkpointing for Distributed Programs," Proceedings of 11th Symp., *On Reliable Distributed Systems*, 155-162, Houston.

[7] R. Koo and S. Toueg, (January-1987), "Checkpointing and Rollback-Recovery for Distributed Systems", *IEEE Transactions on Software Engineering*, SE-13, **1,** 23-31.

[8] S. Neogy, A. Sinha and P. K. Das, (November-2004), "CCUML: A Check Pointing Protocol for Distributed System Processes," TENCON 2004. 2004 IEEE Region 10 Conference, **B**(2), 553-556, Thailand.

[9] B. Gupta, S. Rahimi and Z. Liu, (March-2005), "A New Non-blocking Synchronous Checkpointing Scheme for Distributed Systems," Proc. ISCA 20th Int. Conf. *Computers and Their Applications*, 26-31, New Orleans.

[10] R. Prakash and M. Singhal, (October-1996), "Low-Cost Check Pointing and Failure Recovery in Mobile Computing Systems," *IEEE Transactions on Parallel and Distributed Systems*, **7**(10), 1035-1048.

[11] D. Manivannan and M. Singhal, (December-2002), "Asynchronous Recovery without Using Vector Timestamps," *Journal of Parallel and Distributed Computing*, **62**(12), 1695-1728.

Surender Kumar
Department of Information Technology
HCTM, Kaithal, Haryana, INDIA
E-mail: ssjangra20@rediffmai.com

R. K. Chauhan
Department of Computer Sc. & Applications
K.U. Kurukshetra, Haryana, INDIA
E-mail: rkc_kuk@yahoo.com

Parveen Kumar
Department of Computer Sc. & Engg.
APIIT, Panipat, Haryana, INDIA
E-mail: pk223475@yahoo.com