A Token Ring Minimum Process Checkpointing Algorithm for Distributed Mobile Computing System

P. Kanmani[†], Dr. R. Anitha[††], and R. Ganesan[†††]

[†] Research Scholar, Mother Teresa Women's University, kodaikanal, Tamil Nadu, India.
[††]Director, Department of MCA, K. S. Rangasamy college of Technology, Tiruchengode,Tamil Nadu, India.
[†††] Assistant Professor, Department of MCA, K. S. Rangasamy college of Technology, Tiruchengode,Tamil Nadu, India.

Abstract
Distributed computing poses new challenges in the mobile environment. It has features like high mobility, frequent disconnection and lack of resources such as memory and battery power. Such features make applications running on mobile devices become more susceptible to faults. Checkpointing is an attractive approach for transparently adding fault tolerance to distributed applications without requiring additional programmer efforts. This paper proposes a new non-blocking checkpointing algorithm to tolerate the faults in the mobile computing environment. It is a Min process Token Ring based checkpointing algorithm that reduces the much overheads of the previous non-blocking algorithms. The new algorithm reduces the number of processes taking checkpoints and also diminishes the dependency array passed during the checkpointing process.

Key words:
Checkpointing, Fault Tolerance.

1. Introduction

A distributed system is a collection of processes that communicate with each other by exchanging messages. A mobile computing system is a distributed system where some processes are running on mobile hosts (MHs) that can move. To communicate with MHs, mobile support stations (MSSs) are added. An MSS communicates with other MSSs by wired networks, but it communicates with MHs by wireless networks.

A mobile wireless environment poses challenging problems in designing fault-tolerant systems because of the dynamics of mobility and limited bandwidth available on wireless links. Traditional fault-tolerance schemes cannot be directly applied to these systems. However, fault tolerance is much more important in mobile computing systems since mobile computing systems are more prone to failures. This is because wireless networks have (i) high error rates and more frequent disconnections and (ii) mobile devices are more prone to physical damage, or loss. These problems can be addressed at two levels: the network level and the operating system level. At the network level, the problem can be solved by efficient fault-tolerant channel allocation algorithms and fault-tolerant location management. At the operating system level, the problem can be solved by using checkpointing.

In the Checkpointing Approach, the state of each process in the system is periodically saved on stable storage, which is called checkpoint of the process. To recover from a failure, the system restarts its execution from a previous error-free, consistent global state recorded by the checkpoints of all processes. More specifically, the failed processes are restarted on any available machine and their address spaces restored from their latest checkpoints on stable storage. Other processes may have to rollback to their checkpoints stable storage in order to restore the entire system to consistent state.

Checkpointing Algorithms are classified into two categories blocking [6] and non-blocking algorithms. In the blocking algorithms, all relevant processes in the system are asked to block their computations during checkpointing. Checkpointing includes the time to trace the dependency tree and to save the states of processes on stable storage, which may dramatically reduce the performance of these systems.

Non-blocking algorithms [7] have received considerable attention. In these algorithms, processes need not block during checkpointing by using a checkpointing sequence number to identify inconsistent messages. However, these algorithms assume that a distinguished initiator decides when to take a checkpoint. Therefore, they suffer from the disadvantages of centralized algorithms, such as poor reliability, bottlenecks, etc. Moreover, these algorithms require all processes in the system to take checkpoints during check pointing, even though many of the checkpoints may not be necessary.

2. Related Works

Acharya and Badrinath [1] were the first to present a checkpointing algorithm for mobile computing systems. In their uncoordinated checkpointing algorithm, an MH takes local checkpoint whenever a message reception is preceded by a message sent at that MH. If the send and receive of messages are interleaved, the number of local checkpoints will be equal to half of the number of computation messages, which may degrade the system performance.

Guohong Cao, Mukesh Singhal introduces [2] a new check point approach called as mutable *check point.* A mutable checkpoint is neither a tentative checkpoint nor a permanent checkpoint, but it can be turned into a tentative checkpoint. When a process takes a mutable checkpoint, it does not send checkpoint requests to other processes and it does not need to save the checkpoint on the stable storage. It can save the mutable checkpoint anywhere. Guohong

Manuscript received July 5, 2010
Manuscript revised July 20, 2010

Cao and Mukesh Singhal's mutable checkpoint [2] finds the solution for avalanche effect i.e. the processes in the system recursively asks other process to take checkpoint. But in some cases it will again lead to recursive request of checkpoint.

In [3] P. Kumar, L. Kumar, R.K.Chauhan and V.K.Gupta proposed non-blocking coordinated checkpointing algorithm that require only a minimum number of processes to take checkpoints at any instant of time. They proposed five phase algorithm. In the third and fourth phases the processes take tentative checkpoints and in the fifth phase, the initiator sends a commit or abort message to all the processes.

In [4] the authors have proposed a non-blocking coordinated checkpointing algorithm where in the first phase an initiator sends checkpointing request to all the processes in the system. In the second phase, dependent processes take tentative checkpoint. In the third phase imitator send the commit message to all the processes, if it gets replay from all the processes within the specified time interval and takes its own checkpoint otherwise it sends the abort message.

In [5] the authors proposed a non-blocking algorithm without using any temporary, tentative, or mutable checkpoint with minimum number of processes. In this algorithm, a dependency vector DV is used to store the process history. They are calculating the cost of checkpointing process by using parameters for message passing and broadcasting. The proposed algorithm adapts the method of performance calculation illustrated in their work.

3. Problem Formation

Non-blocking algorithms are based on checkpointing sequence number. But these algorithms are affected by avalanche effect and storage of message history. And also these algorithms are using temporary storage and needs two or more phases to complete the checkpointing process.

The new proposed algorithm reduces these problems by introducing the token ring methodology. In this approach, each and every process involved in the current application has a priority value according to their participations in the current application. The process with the highest priority will be the initiator. After taking its first checkpoint, it sends the checkpoint request as a token to the next process. The process receives the checkpoint requests, takes a checkpoint and passes it to the next lowest priority process. As the proposed algorithm uses token ring, when the token reached the last lowest priority process, it sends back to the initiator. When the token reached back, the highest priority process (i.e. initiator) restarts the next checkpointing process if necessary.

To reduce the number of processes taking checkpoint, a dependency vector is passed as the token in the token ring methodology. The token contains the dependent process number in sorted order. When the highest priority process takes the checkpoint, it updates its flag value as one and creates a dependency vector with the process number depend on it. And it sends the dependency vector as a token to the next lower priority process in the system.

When a token is received by the process, the process updates the dependent vector by deleting its own information and adding only the lower dependent priority processes information. It does not include the information of highest priority process dependent on it. After updating dependent vector, it passes the token to the next lower priority process in the system. When there is no element in the dependent vector the checkpointing process is over. When a process found that there is no process in the dependent token, it sends the finish signal to the initiator.

4. System Model

4.1 System Environment

The distributed computation consists of N processes denoted by P0, P1, P2, _ _ _, PN. The processes communicate with each other by means of message passing. The computation is asynchronous. Each process progresses at its own speed and messages are exchanged through reliable communication channels whose transmission delays are finite but arbitrary. The messages generated by the underlying distributed application, will be referred to as computation messages. Every message in the system is acknowledged. The checkpoint request message will be referred to as token message.

4.2 Problems and Solutions

Initialize all processes in the system with zero flag. The token starts moving from the highest priority process PH to the next and to the next. The process PH initiates the checkpointing process by creating and sending the token. After taking a checkpoint, it resets the flag to 1 and continues the computation. When the process Pi, the next priority process receives a checkpoint request from PH, it takes a checkpoint and sends the token to the next process. The problem may arise because of the two following cases: (i) After taking a checkpoint, if a process receives a message from other processes that has not taken any checkpoint (ii) A process that is not yet participated in the checkpointing process receives a message from other process that has taken a checkpoint already. These problems can be solved by using the following methodology.

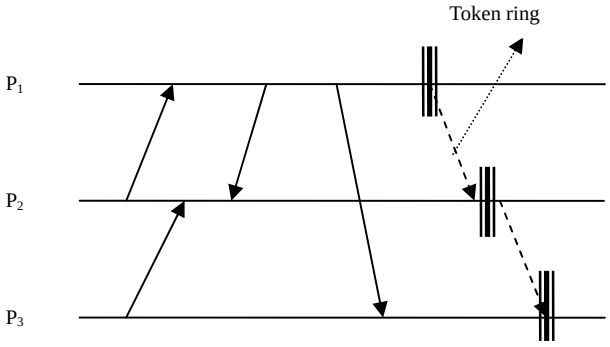

Fig.1 Token Ring in the Best Case

When a process sending a message, each message should piggyback with the current flag value i.e. if it has taken a checkpoint, then it increments the flag value with 1 and sends along with a message.

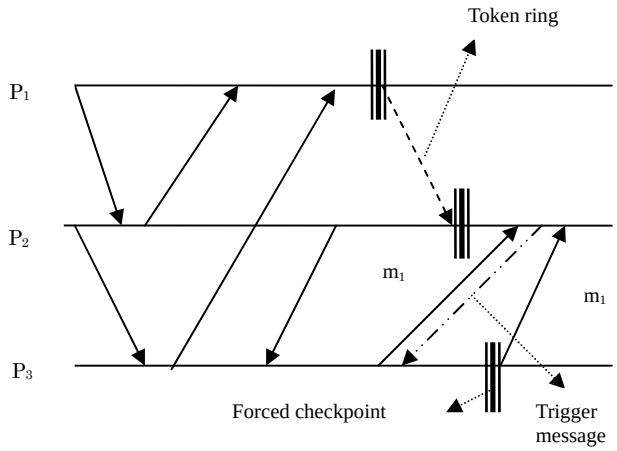

Fig.2 Request with lesser flag value (case i)

When a process receives a message, it checks its flag value with the incoming message flag value. If the values are same, it processes the message (fig.1) otherwise, if its flag value is higher than the message flag value (case i), it sends the checkpointing trigger message to the sending process as in fig.2.

When the process receives the trigger message, it takes a checkpoint, resets the flag value and resends the message with new flag value. Otherwise, like case (ii) if the flag value is less than the incoming message flag value then it takes the checkpoint first and then processes the message as in fig.3.

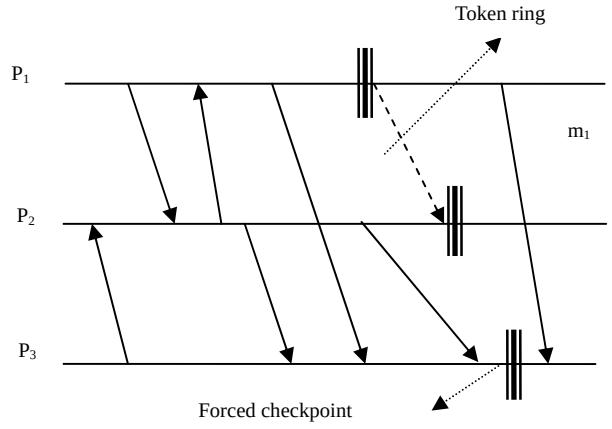

Fig.3 Request with greater flag value (case ii)

When the actual token reaches, the process that takes a forced checkpoint updates only the token up to the forced checkpoint and passes the token to the next process.

Interdependency among the processes may lead to the avalanche effect. One process may send a request to its dependent processes. That process may depend on some other processes. So, it has to send a request to that processes.
This chain never ends. In the proposed algorithm, this avalanche effect is considerably avoided in the initial stage itself because the process can accept the request only if the flag values are equal. After completion of the checkpointing process, the coordinator resets the flag value.

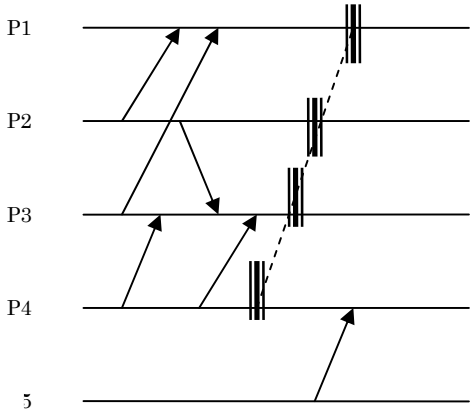

Fig.4 Reduce the number of processes taking checkpoint

5. The Token Ring Non-Blocking Algorithm

At the Coordinator process P_{cor}

Step 1: Initiate the Checkpointing process
Step 2: Creates a token by including the dependent process Detail
Pass the token as a checkpoint request to the next process
Step 3: When F_{npt} signal is received broadcast a message to the process i=1 to n to reset f_{pi} =0

At the process Pi when receiving a token

When a request for checkpoint with token(Dp) is received
Step 1: Check for dependency
Step 2: If dependency exists
If f_{pi} = = 0
(i) Take a checkpoint
(ii) Set f_{pi} =1;
(ii) Update the token Dp by deleting its number and adding lower priority dependent details.
Else If f_{pi} = = 1
Update the Dp as above up to forced checkpoint
Else
If dependency does not exist
Set f_{pi} = 1

Step 3: If Dp is empty
(i) Stop the checkpointing process
(ii) Send F_{npt} to the initiator P_{cor}
Else If Dp is not empty
Send the token to the next lower priority process

At the process Pi when receiving a message with flag value

When a message received from any other process say P_k with a flag value f_k
If f_i = = f_k
Process the message; continue the computation

If $f_i < f_k$
Step 1: Take a forced checkpoint
Step 2: Reset $f_i = 1$;
Step 3: Process the message

If $f_i > f_k$
Step 1: Discord the incoming message
Step 2:Send a trigger message *Trg* to the process Pk;

At the process P_k when receiving *Trg*
When receiving the trigger message from the process P_i as *Trgi*
Step 1: Take a checkpoint;
Step 2: Update the flag value $f_k = 1$;
Step 3: Resend the last message to the process Pi

6. Performance Evaluation and Comparison

The distributed system is simulated in Java. The experimental system includes varies test cases and the following observations are found out

(i) Frequency of forced checkpoints
When the higher priority process are more interacted with lower priority process, the control message is forced, checkpoints are increased

(ii) Blocking time
Proportional to the number of forced checkpoints

(iii) Communication pattern
The frequency of control message is directly proportional to the priority difference between the two communicated processes.

(iv) Length of dependency array passed
Diminished when the token passed from higher priority to lower priority.

The performance of the proposed algorithm is evaluated by calculating the cost needed to complete the checkpointing process. And this cost is compared with the cost of the previous algorithms. The cost of the above stated algorithm is much lower when compared with the other algorithms. The comparison of performance evaluation is presented in the table-1

For the performance calculation the assumed parameters are as follows Consider a distributed system with n+1 process.

Let C_{msg} is the cost of sending message from one process to other process,
N_{min} minimum number of process need to take checkpoint
n_{broad} is the cost of broadcasting a message to all processes in the system.
Tsys - Delay due to system message
Tchk - Delay due to checkpoint storage
N - Number of process involved in checkpointing (When the Time increases, N increases from 0 to N_{min})

In the proposed algorithm the cost in the best case is reduced as $N_{min} * C_{tok}$ and the Blocking time is (Tsys + Tchk) * ($N_{min} - N$). So N_{min} is inversely proportional with delay. The dependency vector is reduced to null.

Table.1 Performance Comparison

	Koo- Toeg	Elnozahy	Mutable	Gupta algorithm	Token Ring
No. of check points	N_{min}	N	N_{min}	N_{min}	N_{min}
Cost	$2* N_{min} * Cmsg$	2* N* Cmsg	$2* N_{min} * C_{msg}$	$N_{min} * C_{msg}$	$N_{min} * C_{tok}$
Blocking time	N_{min} * Tch= in *(Tmsg + Tdata + T disk)	0	2* Tmsg	2* T msg	(Tchk+ Tsys) * ($N_{min} - N$)
Nature of dependency array	Increase with dependency	No dependency array	increased	increased	diminished
Tentative checkpoints	No	No	yes	no	no
Non blocking	yes	yes	yes	yes	yes
Distributed	yes	no	yes	yes	yes

The cost to complete the checkpointing process by using the algorithm [2] is given as $2* N_{min} *C_{msg} + \min(N_{min} *C_{msg}, n_{broad})$ in the best case. In this algorithm first the initiator sends control message to the minimum number of processes. The cost for this is $N_{min} * C_{msg}$. With the acknowledgement message the cost can be calculated as $2 *N_{min} * C_{msg}$. For the commit message at phase II the cost is calculated as $\min (N_{min} * C_{msg} , n_{broad})$.

In [3] the initiator broadcasts dependency vector request to all the n processes and the cost of which is n_{broad}.The initiator receives a vector from the n processes, the cost of which is $n * C_{msg}$. In this way the cost of generating consistent checkpoint is equal to $n * C_{air} + 2 * N_{min} * C_{msg} + 2 * n_{broad}$.

In [4] the initiator broadcasts the checkpoint request to all the processes. The cost of which is n_{broad}. The initiator receives the replay from the n processes the cost of which is $n * C_{msg}$. Finally, the initiator broadcast a commit message to all the processes to convert their temporary checkpoints into permanent ones, the cost of which is $n * C_{msg} + 2* n_{broad}$.

In [5] by using the dependency vector, the checkpoint request is sent to minimum number of processes. So the cost is calculated as $N_{min} *C_{msg}$.

From the Table-1, it is clearly shown that the cost of the proposed algorithm is reduced and the blocking time is reduced than [2] [3] [4] and [5].And also the dependency vector is reduced up to Null. So, the cost of storage used and the cost of sending the dependency vector are reduced as much as possible.

7. Conclusion

This work introduces a new basic idea using token ring concept in designing checkpointing algorithms. The Algorithm outperforms other non-blocking algorithms in many aspects. The present work emphasize on reducing the number of process taking checkpoints, diminish the dependency vector passed during checkpointing process and eliminating the overhead of taking temporary checkpoints. It reduces the number of process taking checkpoints and also the control messages. If the communication pattern is predictable then the algorithm behaves very well with the different priority settings. By giving the lower priority value to the less interactive process the algorithm becomes more suitable and more efficient for mobile environment.

References

[1]. A. Acharya and B.R. Badrinath**,** "Checkpointing Distributed Applications on Mobil Computers," Proc. Third Int'l Conf. Parallel and Distributed Information Systems, Sept. 1994.

[2]. G. Cao and M. Singhal. "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems" *IEEE Trans. Parallel and Distributed System, vol 12, issue 2 pp 157-172, feb 2001.*

[3]. P. Kumar, L. Kumar, R.K. Chauhan and V.K. Gupta, "Non–intrusive minimum process synchronous checkpointing protocol for mobile distributed system", ICPWC 2005, IEEE international conference on personal wireless communications pp 491-495, jan 2005, new Delhi.

[4]. S. Neogy, A, sinha, P.K Das "CCUML: a checkpointing protocol for distributed system processes," TENCON 2004, 2004 IEEE region 10 conference vol B, no 2, pp 553-556, nov 2004, Thailand.

[5]. Bidyut Gupta, Shahram rahimi and Ziping liu "A new high performance checkpointing approach for mobile computing system'IJCSNS International Journal Of Computer Science And Network Security, VOL 6, N05B may 2006.

[6]. R. Koo, S. Toueg, Checkpointing and rollback-recovery for distributed systems, IEEE Trans. Software Eng. 13 (1) (1987) 23–31.

[7]. E.N. Elnozahy, D.B. Johnson, W. Zwaenepoel, The performance of consistent checkpointing, Proc. 11th Symp. on Reliable Distributed Systems, IEEE Press, New York, 1992, pp. 86–95

P. Kanmani M.C.A., M.Phil., she is the Assistant Professor in the Deparment of Computer Science, Thiruvalluvar Government Arts college, Rasipuram, Salem DtTamil Nadu. India.

Dr. R. Anitha M.C.A., Ph.D received her Ph.D in Periyar University salem. .Currently she is the Director of Department of MCA, K. S. Rangasamy College of Technology, Tiruchengode, Namakkal Dt, Tamil Nadu, India

R. Ganesan M.C.A., M.Phil., He is the Assistant Professor in the Department of MCA, K. S. Rangasamy College of Technology, Tiruchengode, Namakkal Dt, Tamil Nadu, India.