

# A High Performance Non-Blocking Checkpointing / Recovery Algorithm For Ring Networks

B. Gupta, N. Mogharreban, S. Rahimi, and A.Vemuri

Computer Science Department  
Southern Illinois University  
Carbondale, IL 62901 USA

*Abstract* - In this paper, we have proposed a new checkpointing / recovery algorithm for ring network architecture. The checkpointing algorithm produces a consistent set of checkpoints in a uni-directional network with the help of few control messages and also avoids the overhead of taking temporary checkpoints unlike most other existing checkpointing algorithms. The number of interrupts to the processes is also less that ensures fast termination of the checkpointing algorithm as well as the application program. The main features of the recovery algorithm are that it is a single step, non-blocking algorithm, with very few interrupts to the processes.

**Keywords:** Ring Network, Checkpointing, Recovery, Non-blocking.

## 1. Introduction

A Distributed System refers to computer systems in multiple locations throughout a network working in a cooperative fashion, with the system at each location not only serving the needs of that location but also able to receive information from other systems, and supply information to other systems within the network. In such an environment inconsistencies in data occur due to faults in system modules. To overcome such inconsistencies different fault-tolerant methods have been employed such as the concept of checkpointing [1]-[7].

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. A local checkpoint is a state of a process saved on stable storage. 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 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) and the consistent set of checkpoints is called consistent global state (CGS).

Checkpointing algorithms are classified broadly into two categories. One is coordinated and the other is uncoordinated. In the later approach processes take local checkpoints independently. Upon the occurrence of a failure; a procedure of rollback-recovery builds a consistent global checkpoint state (CGS) and the system restarts from that state. In the case of coordinated checkpointing an initiator process forces other processes during a failure-free computation to take a local checkpoint each by using control messages. The coordination can be either blocking [3] or non-blocking[9][10]. In a blocking approach all the relevant processes are required to block their application computation during checkpointing and hence system performance is degraded. In non-blocking approach application processes are not blocked when checkpoints are being taken.

Most of the research in the area of coordinated check pointing mainly concentrates on non-blocking. In [5] and [10] authors have discussed non-blocking algorithms for checkpointing in distributed systems. In [4] the authors proposed an algorithm for checkpointing in ring networks; they proposed a coordinated checkpointing algorithm in which processes take checkpoints independent of message pattern. It allows any set of processes in the system to initiate checkpointing. A process need not consider causal dependency generated by the application messages. Due to the special nature of the ring network the scheme does not need to trace dependence at the time of roll back. This algorithm runs faster than the algorithm proposed in [10]. The problem with this algorithm is that the number of control messages transferred is large and there is an overhead of taking a temporary checkpoint and then converting it into a permanent checkpoint. In this work a new checkpointing algorithm is discussed for ring networks where the number of control messages is considerably reduced and the overhead of taking a temporary checkpoint is avoided. And the consistent set of checkpoints is the latest set of checkpoints.

In [4] the proposed algorithm for uni-directional ring networks works in the following way. Consider a ring network consisting of three processes  $P_0$ ,  $P_1$ , and  $P_2$ . Assume that the network is a uni-directional network and hence  $P_0$  can only send messages to  $P_1$  and  $P_1$  to  $P_2$  and  $P_2$  to  $P_0$ . Assume that at any instant of time only one process can start checkpointing algorithm. The presence of the failure  $f$  is considered later in the discussion.

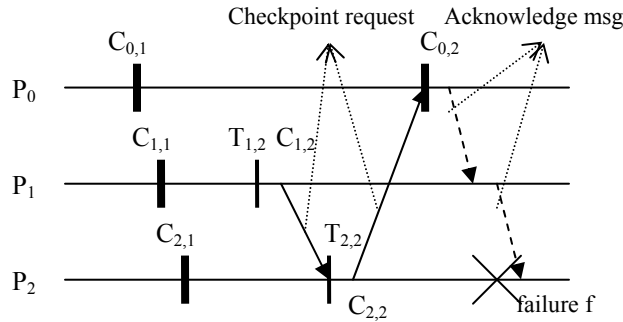


Fig. 1 A system explaining the algorithm proposed in [4]

Let  $C_{0,1}$ ,  $C_{1,1}$ ,  $C_{2,1}$  be the respective permanent checkpoints of  $P_0$ ,  $P_1$ , and  $P_2$  taken during the previous execution of the checkpointing algorithm. Now assume that process  $P_1$  starts the checkpointing algorithm. It takes a temporary checkpoint  $T_{1,2}$  and sends a checkpoint request to its successor process  $P_2$ .  $P_2$  on receiving the checkpoint request takes a temporary checkpoint  $T_{2,2}$  and forwards the checkpoint request to  $P_0$ . Here  $P_0$  sees that it is the predecessor to the checkpoint initiator  $P_1$ ; that is  $P_1$  is the immediate successor of  $P_0$ . Hence it deletes its previous permanent checkpoint  $C_{0,1}$  and takes a new permanent checkpoint  $C_{0,2}$ . It sets the acknowledgement initiator id to its process id 0 and forwards the acknowledge message with the acknowledge initiator id to  $P_1$ .  $P_1$  on receiving the acknowledge message deletes the previous permanent checkpoint  $C_{1,1}$  and converts the temporary checkpoint  $T_{1,2}$  into a permanent checkpoint  $C_{1,2}$  and forwards the acknowledge message to its successor process  $P_2$ .  $P_2$  on receiving the acknowledge message deletes its previous permanent checkpoint  $C_{2,1}$  and converts its temporary checkpoint  $T_{2,2}$  to a permanent checkpoint  $C_{2,2}$ . As it is the predecessor to the acknowledge initiator it terminates the checkpointing algorithm. Clearly it is observed that a number of system messages need to be exchanged among the processes to get a consistent global checkpoint state.

Consider the situation when a failure  $f$  occurs in process  $P_2$  just before it receives the acknowledge message from process  $P_1$ . The recovery algorithm will be initiated by  $P_2$  after it recovers from failure. The recovery is a two step process; first the processes are synchronized on the version of checkpoint number with the help of recovery message; second the processes resume computation from their respective checkpoints to which they rollback with the help of a resume message. We shall show later the algorithm we propose in this paper for recovery is single step and non-blocking.

**Problem Formulation:** In this work we present an algorithm for uni-directional ring networks that does not take any temporary checkpoints. At any instant of time at most two checkpoint states per process are saved in the system which implies that the memory usage by the system is efficient. We show that the time taken to complete a single execution of the checkpointing algorithm is much less when compared to the algorithm proposed in [4]. The proposed recovery algorithm is a single step non-blocking algorithm unlike in [4]. Besides, the number of control messages used in our approach and the number of interrupts are much less when compared to the algorithm in [4].

## 2. System model

We consider the uni-directional ring network architecture. It consists of  $n$  processes  $P_0, P_1, P_2, \dots, P_{n-1}$ . In this system processes do not share common memory and communicate via messages between each other. The processes in the system are named sequentially. In a unidirectional ring system  $i^{\text{th}}$  process can directly send a message to  $j^{\text{th}}$  process if and only if  $j = (i+1) \bmod n$ . That is  $j^{\text{th}}$  process is the immediate successor to  $i^{\text{th}}$  process if and only if  $j = (i+1) \bmod n$ . The communication channel is assumed to be first in first out (FIFO).

### 3. Data Structures

Consider a distributed system of  $n$  processes  $P_0, P_1, P_2, \dots, P_{n-1}$  involved in the execution of a distributed algorithm on a uni-directional ring network. In a uni-directional ring network a process  $P_i$  can send an application or control message only to its successor process  $P_j$  where  $j = (i+1) \bmod n$ .

We use two kinds of control messages in our algorithms. One is a checkpoint request  $\langle C_p, i \rangle$  which as in [4] is sent by an initiator process to initiate the checkpoint scheme to its successor process where  $i$  is the process id of the initiator process and the second control message is the cancellation message  $\langle C_c, i \rangle$  sent by the initiator process  $P_i$  to its successor process asking to delete its checkpoint taken during the current execution.

### 4. An Illustration

We illustrate our checkpointing scheme with an example. Consider the following scenario of a distributed system of four processes  $P_0, P_1, P_2$ , and  $P_3$  on a uni-directional ring network architecture as shown in Fig. 1a. Suppose  $P_2$  decides to initiate the checkpointing algorithm. It sends the checkpoint request  $\langle C_p, 2 \rangle$  to its successor process  $P_3$  and then take its checkpoint  $C_{2,1}$ . The steps of sending the request and taking the checkpoint are atomic. When process  $P_3$  receives the checkpoint request from  $P_2$  it forwards the checkpoint request to its immediate successor process that is  $P_0$  and takes its checkpoint  $C_{3,1}$ . On receiving the checkpoint request  $P_0$  forwards the checkpoint request to  $P_1$  and takes its checkpoint  $C_{0,1}$ .  $P_1$  forwards the checkpoint request to  $P_2$  and takes a checkpoint  $C_{1,1}$ .  $P_2$  sees that it has got back the checkpoint request message initiated by it and therefore terminates the algorithm.

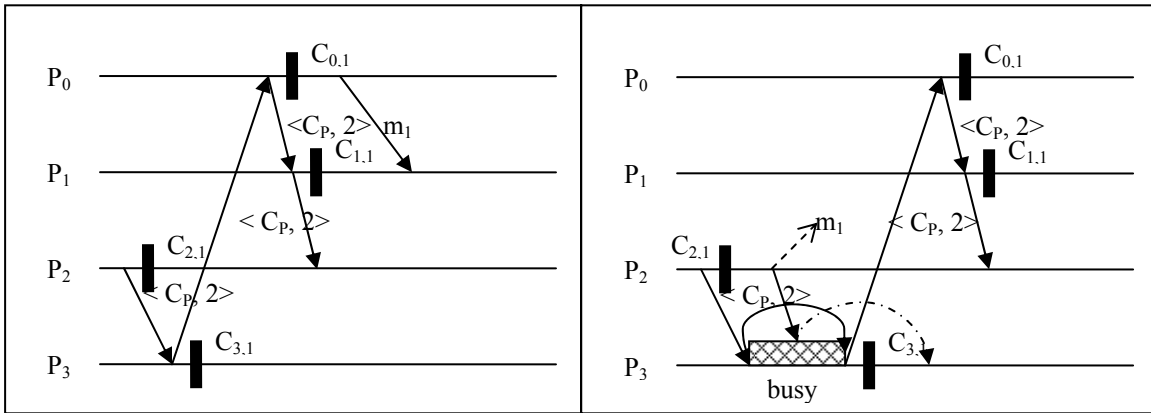


Fig. 1a No Process is busy

Fig. 1b Process  $P_3$  is busy

Note that in this example after taking the checkpoint  $C_{0,1}$  process  $P_0$  sends an application message  $m_1$  to  $P_1$ . Since checkpoint request is first sent before the application message and our system follows First in First out (FIFO) communication channel therefore  $m_1$  cannot reach  $P_2$  before checkpoint request  $\langle C_p, 2 \rangle$ . Hence the message  $m_1$  can never be orphan. Consider the following typical situation.

Suppose that process  $P_3$  is busy when it receives the checkpoint request  $\langle C_p, 2 \rangle$  and cannot process the request. It is shown in Fig. 1b. In such a situation all the messages are queued up at process  $P_3$ . When process  $P_3$  finishes its current execution it starts processing the queued requests sequentially. Hence in this case first the checkpoint request is processed and  $P_3$  forwards the request to its successor  $P_0$  and takes its checkpoint and then processes the application message  $m_1$ .  $P_0$  and  $P_1$  execute the checkpointing algorithm. When the control message finally reaches process  $P_2$  the algorithm is terminated. The algorithm for the uni-directional ring network is given below.

#### 4.1 Algorithm A1

The responsibilities of initiator process and all the other processes are stated below:

Initiator process  $P_i$

- Step 1: Send a checkpoint request  $\langle C_p, i \rangle$  to successor process;
- Step 2: take a checkpoint replacing the previous checkpoint;
- Step 3: continue with normal execution;

Step 4: if checkpoint request  $\langle C_p, i \rangle$  has come back terminate the algorithm;

Process  $P_j$

```

if it receives checkpoint request  $\langle C_p, i \rangle$ 
  if it is busy
    it finishes current computation;
    processes the checkpoint request found in its queue of incoming messages
    forward the checkpoint request  $\langle C_p, i \rangle$  to successor process;
    take a checkpoint replacing the previous checkpoint;
    continue normal execution;
  else
    forward the checkpoint request  $\langle C_p, i \rangle$  to successor process;
    take a checkpoint replacing the previous checkpoint;
    continue normal execution;

```

We propose another algorithm for the uni-directional ring network which works as follows. When any process is busy, instead of waiting the initiator times out and cancels all the checkpoints taken. Later the same process or any other process can initiate the checkpointing algorithm. In both the approaches the main advantage is that the overhead of taking temporary checkpoints and converting them into permanent checkpoints is absent.

Consider a four process ring network system architecture with processes  $P_0, P_1, P_2,$  and  $P_3$ . The working of the algorithm is similar to the previous algorithm except that when it encounters a busy process the approach changes. Instead of waiting till the process completes its current execution the initiator times out and cancels the checkpoints taken. The algorithm can be restarted after some time by any process.

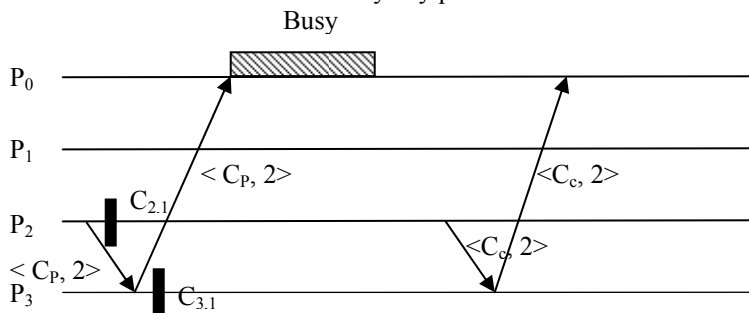


Fig. 1c A special case with cancel request

Here suppose  $P_2$  initiates the checkpointing algorithm sends request to  $P_3$  and takes the checkpoint. Similarly  $P_3$  forwards the request and takes a checkpoint. When the request reaches  $P_0$ ,  $P_0$  is busy with some application and it does not process the request as it cannot interrupt the current execution. Process  $P_2$  waits for a time  $4 \cdot t$  ( we assume that  $t$  is the average message propagation time between a process and its successor. Since Process  $P_2$  has not received the checkpoint request back, it cancels its recently taken checkpoint  $C_{2,1}$  and forwards the checkpoint cancel request with its id to its successor  $P_3$ . On receiving the cancel request  $\langle C_c, 2 \rangle$   $P_3$  first cancels its recently taken checkpoint and forwards the cancel request to its successor  $P_0$ .  $P_0$  did not take a checkpoint in the current execution of algorithm and hence it discards the message.

## 4.2 Algorithm A2

Initiator process  $P_i$

```

Step 1: Send a checkpoint request  $\langle C_p, i \rangle$  to successor process;
Step 2: take a checkpoint;
Step 3: continue with normal execution;
Step 4: if checkpoint request  $\langle C_p, i \rangle$  does not come back after time  $(n-1) \cdot t^*$ 
        send cancel request  $\langle C_c, i \rangle$  to successor process;
      else
        terminate the algorithm;

```

```

Process Pj
  if it receives checkpoint request < Cp, i >
    forward the checkpoint request < Cp, i > to successor process;
    take a checkpoint;
    continue normal execution ;
  else if it receives cancel request < Cc, i >
    if checkpoint has not been taken
      discard the request;
      continue normal execution;
    else
      forward the cancel request < Cc, i > to successor process;
      delete its last checkpoint;
      continue normal execution;

```

**Theorem 1:**

The checkpointing algorithms create a consistent global state of the system.

**Proof:**

According to the system model the communication channel is FIFO. Let us assume that a process P<sub>i</sub> sends a checkpoint request and an application message m to its successor P<sub>((i+1) mod n)</sub>. Both algorithms consider the following possible situations of message transfer:

- (1) message m is sent before the request
- (2) message m is sent after the request

In situation (1), process P<sub>i</sub> takes its own checkpoint after sending the request, so message m cannot be orphan.

We know that the event of sending the request and the event of taking a checkpoint are atomic. Therefore, in situation (2), P<sub>i</sub> first takes its checkpoint and then sends the message m. A process P<sub>j</sub> receives first the request; hence it takes its checkpoint first. So, when message m arrives later at P<sub>j</sub> it cannot be an orphan. Note that in situation (1) a process P<sub>j</sub> receives first the message m followed by the request. So the event of sending m has already been recorded by P<sub>i</sub> in its latest checkpoint and also the event of receiving m is also recorded in the latest checkpoint of P<sub>j</sub>.

Since the above is true for all processes, hence the checkpoints taken by all processes are globally consistent checkpoints. Hence these checkpoints form a consistent global state of the system. ■

## 5. Recovery In A Uni-Directional Ring Network

The recovery process in a uni-directional ring network is explained with an example below. Consider a distributed system of 5 processes P<sub>0</sub>, P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, and P<sub>4</sub> as shown in the Fig. 2. All the processes run the checkpointing algorithm and take their respective consistent checkpoints C<sub>0,1</sub>, C<sub>1,1</sub>, C<sub>2,1</sub>, C<sub>3,1</sub>, and C<sub>4,1</sub>. After the checkpointing algorithm has been terminated processes communicate through messages. Now assume that process P<sub>2</sub> fails.

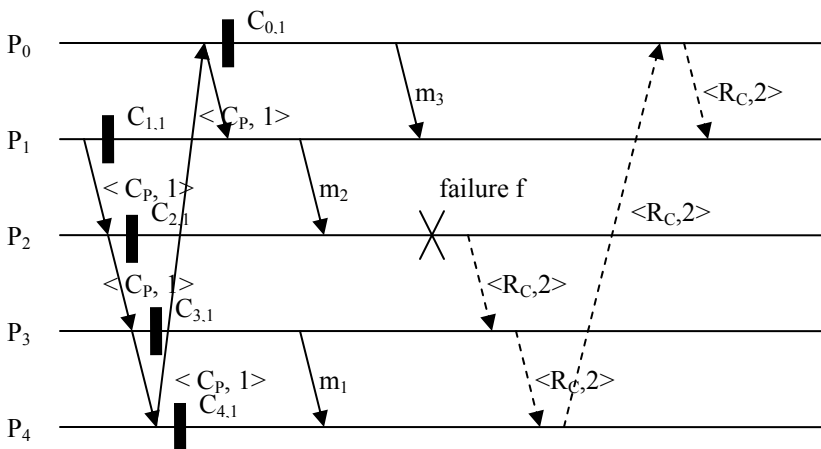


Fig. 2 The recovery approach.

This is shown in Fig. 2. After recovering from failure process  $P_2$  starts the recovery algorithm. It sends the recovery control message  $\langle R_c, 2 \rangle$  with its process id 2 to its successor process  $P_3$ .  $P_3$  on receiving the control message rolls back to the previous checkpoint  $C_{3,1}$  and forwards the message to the next process  $P_4$ . This process continues; and when  $P_1$  receives the control message it rolls back to its checkpoint  $C_{1,1}$  and does not forward the message, because it is the predecessor to the process  $P_2$  which initiated the recovery algorithm. The brief description of the algorithm in a general situation is outlined below.

The recovery algorithm runs in the following way. Whenever a process  $P_i$  initiates the recovery algorithm, it sends a recovery message with its process id to its successor process  $P_{((i+1) \bmod n)}$ . After receiving the control message for recovery each process  $P_j$  rolls back to its previous checkpoint and then forwards the control message to its successor. It then resumes its normal computation. If  $P_j$  is the immediate predecessor to the initiator process  $P_i$ , it terminates the forwarding of the message.

## 6. Advantages

The main advantage of our proposed checkpointing algorithm over the algorithm reported in [4] is that it creates a globally consistent set of checkpoints in less time than that required by the algorithm in [4]. In [4], for an  $n$  process system it takes  $2*(n-1)*t$  time units where  $t$  is the average time taken by each process to send a message to its successor process. The checkpointing algorithm proposed in this paper will find a consistent set of checkpoints in  $n*t$  time units.

Besides, when the number of control messages used in our algorithm and in the algorithm [4] are compared, it is observed that for an  $n$  process system the algorithm in [4] requires  $n-1$  checkpoint requests, and  $n-1$  acknowledge messages. Our proposed algorithm requires  $n$  control messages to complete the execution of the algorithm. The graphs in Fig.3a and Fig.3b clearly show the advantages of our proposed algorithm compared to the algorithm in [4].

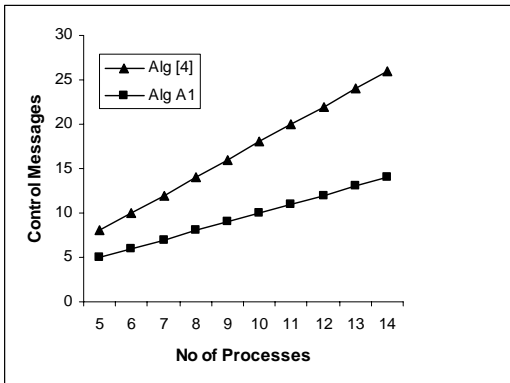


Fig. 3a Comparison of control messages

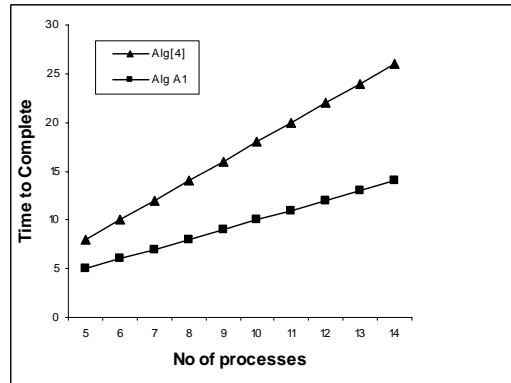


Fig. 3b Comparison of the execution times

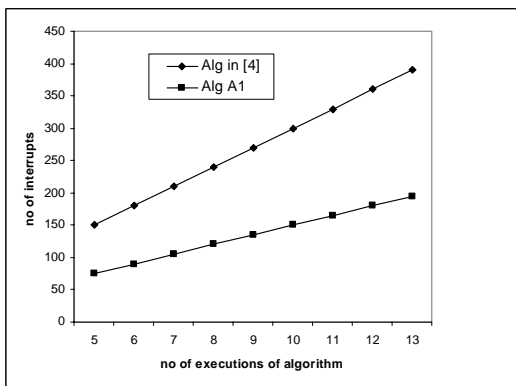


Fig. 3c Comparison of number of interrupts

When we compare our proposed algorithm with the one in [4], we observe the following. The advantage of our algorithm is that it is a one-step non-blocking algorithm. Once a process rolls back to its previous state it forwards the request to its successor and then it continues with normal execution. The algorithm in [4] follows a two-step approach, where first a recovery message is sent to establish synchronization over same checkpoint number and then a resume message to restart computation from the point to which it rollback. Clearly as a result the number of interrupts is much larger when compared with our recovery algorithm. It is shown in Fig. 3c.

## 7. Conclusion

In this paper, we have presented a new non-blocking algorithm for checkpointing in uni-directional ring networks. The important features of the proposed checkpointing algorithm are; it uses very few control messages to determine a consistent global state of checkpoints. There is no overhead of taking temporary checkpoints unlike in [4]. The recovery algorithm also requires very few control messages when compared to the algorithm proposed in [4]. Moreover the recovery algorithm proposed here is a single step algorithm. Also, since our checkpointing algorithm does not take any temporary checkpoints, therefore it is faster than the one in [4]. Fewer number of interrupts used in our present approach also contributes to the speed of the execution of the algorithms.

## 8. References

- [1] Y-M Wang, "Consistent Global Checkpoints that contain a given set of Local Checkpoints", *IEEE Transactions on Computers*, Vol.46, No.4, pp.456-468, 1997.
- [2] M. Singhal and N. G. Shivaratri, *Advanced Concepts in Operating Systems*, McGraw-Hill, 1994.
- [3] R. Koo and S. Toueg, "Checkpointing and Rollback-Recovery for Distributed Systems", *IEEE Transactions on Software Engineering*, SE-13, (1), pp. 23-31, 1987.
- [4] P.S. Mandal and K. Mukhopadhyaya, "Concurrent checkpoint initiation and recovery algorithms on asynchronous ring networks", *Journal of Parallel and Distributed Computing*, Vol. 64, Issue 5, pp. 649-661, May 2004.
- [5] G. Cao and M. Singhal, "On Coordinated Checkpointing in Distributed Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 9, No.12, pp. 1213-1225, 1998.
- [6] D. Manivannan and M. Singhal, "Quasi-Synchronous Checkpointing: Models, Characterization, and Classification", *IEEE Transactions on Parallel and Distributed Systems*, Vol.10, pp. 703-713, 1999.
- [7] G. Cao and M. Singhal, "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol.12, pp. 157-172, 2001.
- [8] E. N. Elnozahy, D. B. Johnson, and W. Zwaenepoel, "The Performance of Consistent Checkpointing", Proc. 11<sup>th</sup> Symposium on Reliable Distributed Systems, pp. 86-95, Oct. 1992.
- [9] L. M. Silva and J. G. Silva, "Global Checkpointing for Distributed Programs", Proc. 11<sup>th</sup> Symposium on Reliable Distributed Systems, pp. 155-162, Oct. 1992.
- [10] R.Prakash and M. Singhal, "Maximum global snapshot with concurrent initiators", Proc. 6<sup>th</sup> IEEE Symposium of Parallel and Distributed Processing, October 1994.