computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

CONSISTENCY OF DISTRIBUTED SYSTEM WITH ACTIVE INITIATOR PROCESS WITHOUT USELESS CHECKPOINTS

Gopalan N.P., Nagarajan K.

Dept. of Computer Science and Engg, National Institute of Tech., Tiruchirappalli, Tamilnadu, India – 620 015
{gopalan, csk0303}@nitt.edu

Abstract: *Checkpointing mechanism is the one of the best attractive approach for providing software fault tolerance in distributed message passing systems. This paper aims to implement a distributed checkpointing technique, which eliminates the drawbacks of the centralized approach like "domino effect", "useless checkpoint" (checkpoints that do not contribute to global consistency), and "hidden and zigzag" dependencies. The proposed checkpointing protocol has a checkpoint initiator, but, coordination among the local checkpoints is done in a distributed fashion. This guaranty that no message would be lost in case of failure occurs, has been maintained in this work by exchange of information among the processes. However, there is no central checkpoint initiator, but each of the processes takes turn to act as an initiator. Processes take local checkpoints only after being notified by the initiator. The processes synchronize their activities of the current checkpointing interval before finally committing their checkpoints. Thus, the checkpointing pattern described in this paper takes only those checkpoints that will contribute to the consistent global snapshot thereby eliminating the number of useless checkpoints.*

Keywords: *Asynchronous distributed system, software fault tolerance, consistent global checkpoint, useless checkpoint, checkpointing Interval, initiator process and consistent state.*

1. INTRODUCTION

In a distributed system a finite set of processes interact to achieve a common goal. The processes are synchronized by exchanging messages over a reliable communication network, in the absence of a global clock. The delay in the message transfer is finite and unpredictable. These are the characteristics of the well-known *"asynchronous distributed systems"* [6]. When computation is extensive, the possibility of process failures may be on the rise. Many techniques have been developed to increase reliability of the system making them highly available. One of them is *rollback-recovery*; which is the process of undoing the computations from the most recent *checkpoint* (CP) and resuming processing [12]. The CP is a snapshot of the state of a process saved on nonvolatile storage.

A *global checkpoint* (GCP) of a system is a set of local CPs one from each process [19]. During message transmission, if a receiver is down, then the message becomes *missing* and if the sender crashes, the message is termed as *orphan*. A system can have *consistent global checkpoint* (CGCP) while these two types of messages are absent [7]. A process, designated as *CP-initiator* takes care of CGCPs in the present work and each process is made to take turn and act as the initiator. Generally, processes take local CPs after being notified by the initiator except in certain cases. The processes synchronize their activities before committing their CPs. This removes inconsistency, if any, when CPs are committed [22]. In the present study disallows the formation of *zigzag paths* and *cycles* [9] by this strategy as the checkpointing pattern makes use of only that CPs that will contribute to a CGCP. If the i^{th} set of the CPs can be proved to be consistent, then in case of recovery the system has to rollback only up to this state [5].

The processes do not post their status information along with computation messages but update-their status whenever a message is sent or received. Whenever process fails, all its dependent processes may not necessarily rollback to the most recent consistent CP. When processes resume after roll-back, they get duplicate messages from other processes using suitable request, inspect stable storage log information for the already received messages from other processes. Alternatively, duplicate messages can be received or sent to other processes using communication primitives.

Dependent processes are identified using the information piggy-backed with the sent and received messages. The CPs from every pair of dependent

processes is used to construct a GCP for each group of dependent processes and this is found to be consistent in the present study. One of the processes in the system acts as CP-initiator and is considered to be reliable until CGCP is established. For all dependent processes after which the next designated process takes over its jobs and a complex election algorithms should the CP-initiator be prone for downtimes. Complex and complicated election algorithm may be thought of in this contest. The rest of the paper is organized as follows: Section 2 throws light on some related works in this area. Section 3 describes the system model and its notations. Section 4 discusses in details of the checkpointing algorithm with a proof and section 5 shows the result of simulation. Section 6 provides the comparison with the earlier work of the message passing systems with different checkpointing protocols and section 7 concludes the paper.

2. RELATED WORK

With reference to Chandy and Lamport [5], Wang et. al [21], Tsai and Kuo [20] states that "*A GCP 'M' is consistent if no message is sent after a CP of 'M' and received before another CP of 'M'* ". Following these observations we regard consistency as the scenario where if a sender *'S'* sends a message *'m'* before it has taken its i^{th} CP, then message *'m'* must have to be received by a receiver *'R'* before the receiver has taken its i^{th} CP. A message will be termed '*missing'* if its send is recorded but receipt is not and otherwise it is termed as '*orphan'* [18]. Suppose a node fails after taking its i^{th} CP, it is desirable that the system in such a scenario should rollback to the last (i^{th}) saved state and resume execution from there. If a system can ensure that there is no *missing* or *orphan* message in the concerned i^{th} GCP, then the set of all the i^{th} CPs taken by its constituent processes is bound to be consistent. Unlike the approach that should exist in a distributed system, Kalaiselvi and Rajaraman [10] have kept record at the message sending end and at the message receiving end. A CP coordinator matches the log it gets from all the processes at each checkpointing time. The present system also keeps records of messages sent and received in each process but the log is matched in a distributed fashion. Due to disparity in speed or congestion in the network, a message belonging to $(i+1)^{th}$ checkpointing interval (CPI) may reach its receiver who has not yet taken its i^{th} CP. Such a message is discarded in [3, 4] and sender retransmits it. Another method of dealing with such messages is to prevent their occurrences by compelling the sender to wait for a certain time before sending a message after any checkpoint [1]. The present work discards such a message by adopting a technique in receiving whereas in another approach [7] any process refrains from sending during the interval between the receipt of CP initiation message and completion of committing that CP. Distributed systems that use the recovery block approach [6, 10] and have a common time base may estimate a time by which the participating processes would take acceptance test. These estimated instants form the pseudo point times as described in [14]. The disadvantages of such a scheme are more than one, like, fast processes may have to wait for slow processes to catch up and other fault tolerance mechanisms like time out may be required. In [12, 14] the authors have analyzed checkpoints taken in a distributed system having loosely synchronized clocks [13, 16]. No special synchronization messages have been used in those methods but the existing clock synchronization messages were utilized. The work described in [1, 3] however, allows processes to take CPs on one's own and then a CGCP is constructed from the set of local CPs. The drawback of the method is that useless CPs can't be avoided. The approach taken by Strom et al. in [17] does not maintain a CGCP at all times but has to save enough information to construct such a CP when need arises. So, this requires logging of messages. Contrary to the present checkpointing protocol, Prakash et. al. [15] presents minimal snapshot collection protocol where dependency is calculated during checkpointing also.

3. SYSTEM MODEL AND NOTATIONS

The system consists of *'n'* processors, P_0, P_1, …, P_{n-1}. Let C_k^i denote the i^{th} CP of k^{th} process (for example, the k^{th} process, initial-CP is C_k^0 (for i=0), first-CP is C_k^1 (for i=1), second-CP is C_k^2 (for i=2) and so on). The CPI is the time interval between any two consecutive CPs and its i^{th} CPI ends at C_k^i. The k^{th} process starts its execution at C_k^0 (k = 0,k = 1,…,k = n-1). To begin with a process, say P_0, initiates checkpointing procedure. The next checkpoint initiation is done by P_1 and so on and forth. Further, the initiation of checkpointing at regular intervals is done by processes. Asynchronous communication has been assumed among the processes. Acknowledgement and time-out are part of the communication protocols.

4. ALGORITHMS AND DESCRIPTIONS

The algorithm has a CP initiator which sends an implicit CP synchronization messages. The initiator sends the initiation message to all other processes with the information such as: (i) The number of

messages sent to processes in the current CPI and (ii) The number of messages received from processes in the current CPI. It must be mentioned here that the additional information regarding messages would not be sent during the initial CP since it is taken just after the system has been initialized and hence it is assumed that communication among processes has not yet started. The information means that if P_k has sent a total of two messages to P_j in the current CPI, then P_k would write 2 as number of messages and j as process id (PID) as part of the first information. Similarly if P_j has indeed received all the two messages from P_k it would write 2 as number of messages and k as PID as part of the second information. P_j checks whether the total number of messages sent by P_k matches with that received by P_j. If the answer is positive, P_j takes the checkpoint. If not, then it waits for the undeceived message(s) and takes the CP after receiving it (them). During this time only those messages are received for which P_j is waiting and any unwanted messages is discarded.

The list of variables used in the algorithm is described as follows:

Own_Pid:	PID of the process responding to the communication.
Initiator:	PID of CP initiator.
CP_Seq:	CP sequence number, initially 0.
CP_Indx:	Index of the current CPI, initially 0
$MST_i[j]$:	Values denote the number of messages sent by the process i to process j in the current CPI. (Example, $MST_2[5] = 3$, specifies that the process P_2 has sent 3 messages to P_5 in the current CPI).
$MRF_i[j]$:	Values denote the number of messages received by the process i from process j in the current CPI. (Example, $MST_0[2] = 1$, specifies that the process P_0 has received 1 message from P_2 in the current CPI).

CP_Consistency is the subroutine and Send_CP_Req, Receive_CP_Req, Send_PSI and Receive_PSI are the set for communication primitives used by the proposed algorithms.

Algorithm Checkpoint

```
{
1. if (Own_Pid = Initiator) then {
2.   if (CP_Seq = 0) then {
3.      Take a CP and increment CP_Seq by 1;
4.      ∀PID: (0≤ PID ≤ n-1 and PID ≠ Own_Pid)
        Send_CP_Req (CP_Seq, CP_Indx);}
5.   else {  // CP_Seq ≠ 0
6.      Take a tentative CP and increment CP_Seq
        and CP_Indx by 1;
7.    ∀PID: (0≤ PID ≤ n-1 and PID ≠ Own_Pid)
      Send_CP_Req (CP_Seq, CP_Indx) and
      Send_PSI (CP_Seq, CP_Indx, MST, MRF);}}
8. else { // PID ≠ Initiator
9.    if(CP_Seq = 0) then {
10.      Receive_CP_Req (Rcvd_CP_Seq,
        Rcvd_CP_Indx) from process P0;
11.     Repeat line 3.;}
12.   else { // CP_Seq ≠ 0
13.    Receive_CP_Req (Rcvd_CP_Seq,
       Rcvd_CP_Indx) from the initiator;
14.    ∀PID: (0≤ PID ≤ n-1 and PID ≠ Own_Pid) do
       {Receive_PSI (Rcvd_CP_Seq,
       Rcvd_CP_Indx, Rcvd_MST, Rcvd_MRF);
15.    Send_PSI (CP_Seq, CP_Indx, MST, MRF)
          check CP_Consistency (Rcvd_MST,
          Rcvd_MRF);}
16.    Receive_PSI (Rcvd_CP_Seq, Rcvd_CP_Indx,
       Rcvd_MST, Rcvd_MRF);
17.    Check CP_Consistency (Rcvd_MST,
       Rcvd_MRF);}}
18. }
```

Algorithm CP_Consistency (MST, MRF)

```
1. {
2.  for(i=0;i≤n-1;i++)
3.    for(j=0;i≤n-1;j++){
4.     if((i≠j){
5.      if((MSTi[j] = MRFj[i])){
6.           Convert a tentative CP of i^th and j^th
             processes are permanent;
7.           Increment CP_Seq by 1 and reset CP_Indx
             to initial value;}
8.      else if (j = Own_Pid){
9.       Receive_CP_Req (Rcvd_CP_Seq,
         Rcvd_CP_Indx) from the initiator;
10.      ∀PID: (0≤ PID ≤ n-1 and PID ≠ Own_Pid) do
         Receive_PSI(Rcvd_CP_Seq,Rcvd_CP_Indx,
         Rcvd_MST, Rcvd_MRF);
         Check CP_Consistency (Rcvd_MST,
         Rcvd_MRF);}
11.    else{ // MSTi[j] ≠ MRFj[i]
12.      Discard the tentative CP;
13.      Expand the CP_Indx and Decrement
         CP_Seq by 1;}}}
14.  }
```

The *Checkpoint algorithm* works as follows: The initial checkpoint is taken after system initialization by the CP-initiator (Cf. lines 1-4) and other processes (Cf. lines 9-11). For any other CPs, the initiator first sends a '*CP-request'* along with its process status information within the current CPI (Cf. lines 6 and 7). Other than the initiator processes

receive and send their status information to the initiator and exchanges among them, which is described in lines 12-15. Some slow process may wait and receiving status information in line 16. After it receives status information from all others it goes on to check system consistency in line 17. When the system is consistent, commit the tentative CP (Cf. lines 5-7); otherwise discard the tentative CP (Cf. lines 11-14) using CP_Consistency subroutine. This is demonstrated using Boldoni et.al. algorithm in [2] and Lamport '*happened before relation'* [21] as follows:

Theorem 1

A GCP $\{C_k^i\}$ is consistent only when all pair of local CPs $\{C_k^i, \{C_{k+1}^i\}\}$ is consistent (where, $\forall k, 0 \le k \le n-1$) and the CP taken by the algorithm forms a CGCP.

Proof: This theorem is proved by contradiction.

Let us consider the checkpoints form an inconsistent GCP. Then there should be a checkpoint C_s^i that happened before [11] another checkpoint C_r^i. This implies that, the two scenarios were obtained as follows:

1. There should be at least a message 'm' sent by the process P_s after C_s^i but received by the process P_r before C_r^i.
2. There should be at least a message 'm' sent by the process P_s before C_s^i but received by the process P_r after C_r^i.

Therefore, the pair (C_s^i, C_r^i) in not a consistent CP and it is not a part of CGCP. This can be proved in the following way:

Case 1:

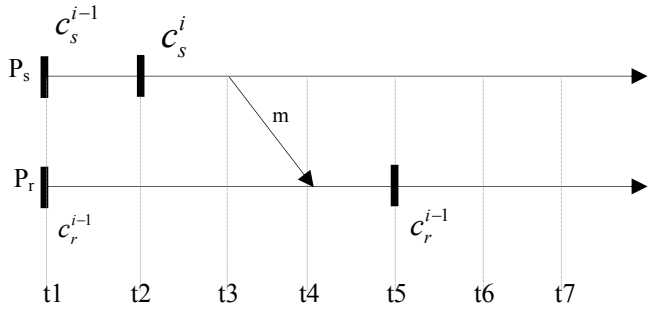

Fig. 1

Let us consider figure 1 and a fault-free scenario where messages reach destinations correctly.

Assumptions:

1. Message 'm' not recorded sent.
2. Message 'm' recorded received.
3. C_s^{i-1} and C_r^{i-1} are the CGCPs.
4. C_s^i and C_r^i are the tentative local CP (initially not recorded in the stable storage) taken by the processes P_s and P_r respectively after the CP request message received.

The following scenario is observed:

i. Message 'm' is sent at t3 and tentative C_s^i taken by the process P_s at t2. (t2 < t3 by assumption 1)
ii. Message 'm' is received at t4 and tentative C_r^i taken by the process P_r at t5. (t4 < t5 by assumption 2)
iii. When process P_s takes CP at t2 (by assumption 1 and step i).
    a. P_s has reached line 17 of algorithm via lines 12-16.
    b. P_s has checked its consistency with other (n-1) processes including P_r using algorithm CP_Consistency.
iv. In line 15 P_s sends its status and P_r receives it in line 16.
    a. P_r check the system consistency in lines 1-5 using algorithm CP_Consistency and no discrepancies are noted.
    b. P_r reaches line 6 and takes C_r^i by step ii. Therefore, violating assumption 2 and scenario ii.
    c. Message 'm' reaches P_r and eventually gets rejected using lines 8-14 of algorithm CP_Consistency (by step i).
    d. Step iv. (b and c) contradicts assumption 2.

Therefore, there can't be any message 'm' that is not recorded sent but recorded received in the same GCP.

Case 2:

Let us consider Figure 2.

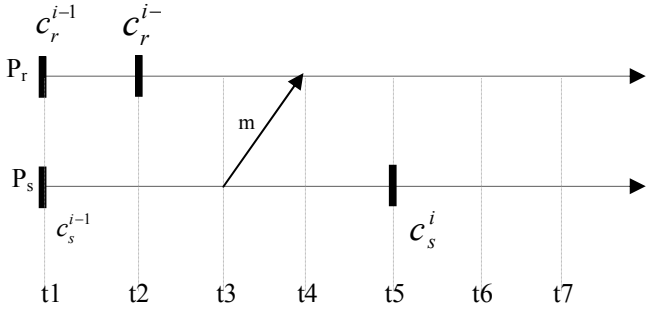

Fig. 2

Assumptions:

1. Message 'm' is recorded sent.
2. Message 'm' is not recorded received.
3. c_s^{i-1} and c_r^{i-1} are the CGCPs.
4. c_s^i and c_r^{i-1} are the tentative local CP (initially not recorded in the stable storage) taken by the processes P_s and P_r respectively after the CP request message received.

The following scenario is observed:

i. Message 'm' is sent at t3 and tentative c_s^i taken by the process P_s at t5. (t3< t5 by assumption 1)
ii. Message 'm' is received at t4 and tentative c_r^{i-1} taken by the process P_r at t2. (t2 < t4 by assumption 2)
iii. When process P_s takes CP at t5 (by assumption 1 and step i).
   a. P_s has reached line 17 and recorded sent of message 'm' (by assumption 1 and step i) via lines 12-16 of algorithm checkpoint.
   b. P_s has checked its consistency with other (n-1) processes including P_r using algorithm CP_Consistency.
iv. Similarly, when P_r takes CP at t2.
   a. P_r reaches line 6 using algorithm CP_Consistency.
   b. P_r has checked its consistency with other (n-1) processes including P_s using lines 1-5 of algorithm CP_Consistency.
   c. P_r finds that message 'm' from P_s is yet to be received by it. (by step i).
   d. P_r checks the system consistency in line 16 via lines 12-16. But the message 'm' is not actually received in line 16 of algorithm Checkpoint.
   e. Therefore, P_r can't reach line 17 and can't take CP.
   f. Step iv.e. contradicts assumption 2.

Therefore, there can't be any message 'm' that is recorded 'sent' but not recorded 'received' in the same GCP protocol.

5. PERFORMANCE EVALUATION

The experiments were performed on a cluster of PCs under Linux 2.4.18. The cluster consists of 8 nodes connected by a 100 MBPS Ethernet and equipped with AMD processors running at 1.2GHz with 128KB catch, 256MB of main memory and 20GB of stable storage.

A Dense matrix multiplication (MM) application [8] is used for the performance evaluation of the proposed algorithm. The program implementations use the LAM/MPI version 1.2.5 and the program was compiled using the GNU GCC version 2.96. The same application is executed using distributed check pointing (DCP), coordinated check pointing (CCP) and communication induced check pointing (CICP) protocols for the comparison study.

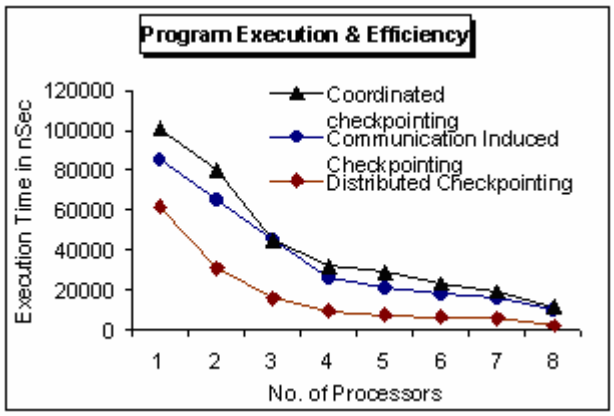

Fig. 3 - Execution Time Vs No. of Processors

The Figure 3 illustrates the execution time under different CP protocols performance, when the numbers of processors are varied in the cluster. With the increasing synchronization overhead and failures with the increase in number of processors gets degraded in CCP and CICP. Hence, the execution times are found to be higher by about (38%, 25.04%), (69.46%, 47.34%), (147.25 %, 69.6%), (158.7%, 103.04%) and (196.4%, 143.6%) than those observed in the present DCP model using 1,2,3,4 and 8 processors in action.

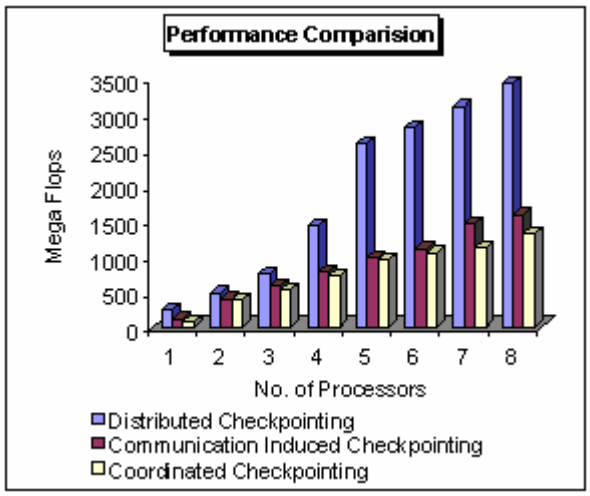

Fig. 4 - Performance of different checkpointing protocols

Due to the possible occurrences of domino effect and the dependent processes overheads in CICP and CCP, the failures are increases with proportional to the number of processors. This is clearly described in Figure 4. The performance of DCP variations are qualitatively similar until 3 processors are in action and increases drastically when 4 or more processes. The performance (in Mega Flops) of DCP is (48% and 30%), (54.5% and 50.34%) and (56.19% and 58.59%) higher during compression with CCP and CICP protocols when 1, 4 and 8 processors are in action.

6. COMPARISON WITH THE EARLIER WORK

Table 1 summarizing the comparison of different checkpointing techniques that are discussed in the survey paper [6, 10]. The following notations are used to compare the present work:

C_{uni}: Cost of sending a message from one process to another process.

C_{brd}: Cost of broadcasting a message to all processes.

T_{ch}: The checkpointing time. i.e, $T_{ch}=T_{msg}+T_{data}+T_{disk}$.

T_{disk} : Delay incurred in saving a checkpoint on the stable storage.

T_{data}: Delay incurred in transferring a checkpoint to the stable storage.

T_{msg}: Delay incurred by system messages during a checkpointing process.

N_{min} : The number of processes that need to take checkpoints.

n: The total number of dependent processes in the system.

N_{tmp}: The number of tentative checkpoints during a checkpointing process.

N_{dep}: The average number of processes on which a process depends.

Table 1. Comparison of various checkpointing approaches

	Uncoordinated Checkpointing	Coordinated Checkpointing	Communication induced Checkpointing	Distributed Checkpointing
Checkpoint Initiator Process	Must be a Separate Process	May or may not be a separate process	May or may not be a separate process	Any of the n processes
Domino effect and useless CP	Possible [3]	Possible [4,14,16]	Possible [12,19,20,21]	Absent
Total Number of CPs	Not possible to say	N_{min}	N_{min}	n
Blocking Time	0	$N_{min}*T_{ch}$	$N_{min}+(T_{ch}*N_{tmp})$	$n*T_{ch}$
Checkpointing Cost	Not possible to say	$3*N_{min}*N_{dep}+C_{brd}$	$N_{min}+N_{dep}+C_{uni}$	$2*(n+C_{uni})$
Total number of messages required for synchronization	System is not set to be synchronized	3 messages. (Request, reply and Acknowledgement)	Not a separate message	2 (Request and Reply)
Vast Network Traffic while synchronization	Towards to the monitor process	Towards to the coordinated process	Piggybacked with the application message	Distributed

7. CONCLUSION

The check-pointing algorithm proposed in this paper constructs consistent distributed checkpoints, without useless checkpoints. Also, the occurrences of missing and orphan messages, hidden and Zigzag paths are avoided. The need for a separate coordinator process doesn't arise.

Further, only a consistent global checkpoint is used and this result in significant performance improvement as the increasing synchronization overhead and failures with the increase in number of processors gets minimized in this approach.

8. REFERENCES

[1] Aurelin, L.Pierre, K.Geraud, C.Franck, "Coordinated checkpoint versus message log for fault tolerant MPI," *Proceeding of the IEEE International Conference on Cluster Computing*, PP: 242 – 250, IEEE CS Press, 1-4 Dec. 2003.

[2] Baldoni, R., J.M.Mostefaoui, A and Raynal M., "A Communication Induced Checkpointing Protocol that Ensures Rollback Dependency Tractability", *IRISA Research Report 1076*, Jan 1997.

[3] Bosilca, G., Bouteiller, A., Cappello, F., Djilali, S., Fedak, G., Germain, C., Herault, T., Lemarinier P., Lodygensky O., Magniette F., Neri V., and Selikhov A., "MPICH-V: Toward a Scalable Fault Tolerant MPI for Volatile Nodes", *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, Baltimore, Maryland, PP: 1 - 18, 2002

[4] Bouteiller Bouteiller, Franck Cappello, Thomas Herault, Krawezik Krawezik, Pierre Lemarinier, Magniette Magniette. "MPICH-V2: a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging," *sc'03*, *ACM/IEEE press*, PP: 25- 42, 2003.

[5] Chandy, M. and Lamport, L., "Distributed snapshots: Determining global states of distributed systems", *ACM Transactions on Computing Systems*, Vol. 3, No. 1, PP: 63-75, Aug. 1985.

[6] Elnozahy, E. N., Alvisi, L., Wang, Y.M., and Johnson D. B., "A survey of rollback-recovery protocols in message-passing systems", *ACM Computing Surveys*, Vol. 34, No. 3, PP: 375–408, 2002.

[7] Gopalan, N.P. and Nagarajan, K., "Self-Refined Fault Tolerance in HPC using Dynamic Dependent Process Groups", *Lecturer Notes in Computer Science (LNCS),* Springer-Verlag, LNCS 3741, pp. 153 – 158, Dec 2005..

[8] Gunnels, J; Lin, C; Morrow, G; and Van de Geijn, R; "Analysis of a Class of Parallel Matrix Multiplication Algorithms," *Proc. Int'l Parallel Processing Symp.*, 1998.

[9] Jichiang Tsai, "On Properties of RDT Communication-Induced Checkpointing Protocols", *IEEE Transactions on Parallel and Distributed Systems*, Volume 14, Issue 8, Pages: 755 – 764, August 2003.

[10] Kalaiselvi, S. and Rajaraman, V, "A survey of rollback and recovery strategies for computer programs", *IEEE Transaction on Computer*, Vol. 25: PP 489–510, October 2000.

[11] Lamport, L., "Time, Clock and the ordering of events in a Distributed System", *Communications of ACM*, 21(7): 558-567, 1978.

[12] Manivannan, D.; Netzer, R.H.B.; Singhal, M.; "Finding Consistent Global Checkpoints in a Distributed Computation", *IEEE Trans. On Parallel & Distributed Systems*, Vol. 8, No.6, June 1997, PP 623 – 627.

[13] Manivannan, D., "Quasi-Synchronous Checkpointing: Models, Characterization and classifications", *IEEE Trans. On Parallel and Distributed Systems*, Vol. 10. No. 7, July 1999, PP 703 –713.

[14] Neogy, S. Sinha, A; Das, P.K., "Finding Consistent Checkpoints in a Distributed System with Synchronized Clocks", *IASTED International Conference on Applied Informatics* AI -2001, February 19 – 22, Australia.

[15] Prakash, R; Singhal,M; "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems", *IEEE Trans. On Parallel and Distributed System*, Vol. 7, No. 10, PP 1035-1048, October 1996.

[16] Sinha, A; Das, P.K.; Basu, D.; "Implementation and Timing analysis of Clocks Synchronization on a Transporters based replicated systems", *Information & Software Technology,* 40(1998), PP 291 –309.

[17] Strom, R.E.; Yemini, S.; "Optimistic Recovery in Distributed Systems", *ACM Trans. On Computer Systems*, Vol. 3. No. 3, Aug. 1985, PP 204 –226.

[18] Tong. Z.; Richard, Y.K. & Tsai, W.T.; "Rollback Recovery in distributed systems using loosely synchronized clocks", *IEEE Trans. On Parallel and Distributed Systems*, Vol. 3. No.2, March 1992, PP 246- 251.

[19] Tsai, J.; Kuo, S.; "Theoretical Analysis for Communication Induced Checkpointing protocols with Rollback Recovery Dependency Tractability", *IEEE Trans. On Parallel and Distributed Systems*, Vol. 9, No. 10, Oct. 1998, PP 963-971.

[20] Tsai, J.; Wang, Y.;Kuo, S.; "Evaluation of Domino free communication induced checkpointing protocols", *Information Processing Letters* 69(1999),PP 31- 37.

[21] Wang, Y.M..; Lowary, A; Fuchs, W.K.; "Consistent Global Checkpoint Based on Dependency tracking"*, Information Processing Letters*, Vol. 50, No. 4, 1994, PP 223-230.

[22] Wong, F. and Franklin, M., "Checkpointing in distributed systems," *Journal of Parallel & Distributed Systems*, Vol. 35, No. 1, PP 67–75, May 1996.

Gopalan N.P. *is the Head of the department of Computer Science and Engineering at National Institute of Technology, Trichirapalli, India. He received his Ph.D. from Indian Institute of Science, Bangalore, India, 1983. His research includes Combinatorics, Grid Computing, Distributed & Parallel Computing, Data mining and Algorithms.*

Nagarajan K. *is a Research Scholar of Computer Science and Engineering at National Institute of Technology, Trichirapalli, India. He received his Master of Computer Science and Engg. Degree from Jadavpur University, Kolkatta, 2003. His research includes Distributed Systems, Parallel Processing and Algorithms*