iJRASET
International Journal For Research in
Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2 Issue: IV Month of publication: April 2014

DOI:

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Time Synchronous Adaptive Rollback Recovery Protocol for Mobile Distributed Systems

Monika Nagpal[1], Parveen Kumar[2], Surender Jangra[3]

[1]Research Scholar, Deptt. of CSE, Singhania University Pacheri Bari (Jhunjhunu),(Raj)
[2] Deptt. of CSE, Bharat Institute of Engg. & Tech. Meerut(UP)
[3] Deptt. of IT Engg. HCTM Technical Campus, Kaithal(HRY)

Abstract-- Time can play a important role for determining consistent global state in mobile computing environment in a minimum cost as it does not requires extra coordination message and thus, avoids most causes of overhead. In this paper, an time synchronous adaptive rollback recovery protocol for mobile environment is proposed. The protocol takes minimum number of checkpoints. Proposed protocol also performs very well in the aspects of minimizing the number and size of messages transmitted in the wireless network. It uses time to indirectly synchronise for creating the new consistent state. Therefore, the protocol brings very little overhead to a mobile host with limited resource. Additionally, by taking advantage of reliable timers in MSSs, the time-based rollback recovery protocol can adapt to wide area networks

Keywords— Checkpointing, Global State, Distributed System, Mobile Host, Mobile Support System

I. INTRODUCTION

Checkpointing/rollback recovery is an attractive and popular technique which gives fault tolerance without additional efforts in DSs [11][12]. A checkpoint is a global state of a process stored on stable storage. In a DS, since the processes in the system do not share memory and have not any synchronized clock, a global state of the system is defined as a set of LSs, one from each process. A global state is said to be “consistent” if it contains no orphan message; i.e., a message whose receive event is recorded, but its send event is lost. To recover from failure, the system restarts its execution from a previous CGS saved on the stable storage during fault-free execution.

Adaptive protocol uses time to avoid having to exchange messages during the checkpoint creation. A process saves its state whenever the local timer expires, independently from the other processes. The protocol keeps the various timers roughly synchronized to guarantee that processes' states are stored approximately in the same instant. When the application starts, the protocol sets the timers in all processes with a fixed value, the checkpoint period. The protocol uses a simple re-synchronization mechanism to adjust timers during the application execution. Each process piggybacks in its messages the time interval until the next checkpoint. When a process receives a message, it compares its local interval with the one just received. If the received interval is smaller, the process resets its timer with the received value.

II. RELATED WORK

In [4] Kim and Park’s algorithm the consistency problem is solved by disallowing the Message sending during a period after a time expires. But this makes the checkpointing protocol become a blocking protocol. CHI-YI LIN et. al. algorithm[6] the processes need not induced extra control message to coordinate for producing consistent global checkpointing state. Processes saves its state periodically whenever its local timer expires to indirectly the coordinate the creation of global state except their being obvious orphan and in-transit messages because timer are not well synchronized. These potential orphan or in-transit messages are consistent usually. They become true orphan or in-transit messages only in some special period.

In [9] Men Chaoguang’s algorithm a two phase technology is used to handle potential inconsistent issues that may arise in Time-based algorithm. Assume that D is maximum deviation between the checkpoint timer of two processes and T is checkpoint period. MD is maximum deviation of two processes where $MD = D + 2n\rho T$. This is blocking time in which a process can not send or receive messages otherwise inconsistency may arise due to orphan or in-transit messages the maximum and minimum message propagation delays are tdmax and tdmin. The messages sent, MD + tdmax time units before taking checkpoint may become in-transit messages. There are two kind of messages that can result in system inconsistencies. As shown in Fig.1 m1, m4 are potential

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

inconsistent message because the processes takes checkpoints in different time , messages m2 is an obvious in-transit messages and m2 is an obvious orphan messages. In the period T1 to T2, m1 become orphan temporarily. If fault occur during T1 to T2 the system cannot recover to consistent state otherwise m1 become a normal message.

In two phase technology two timers, Timer_ckp and Timer_pmt, are used to solve potential inconsistent issues. Whenever Timer_ckp expires, tentative checkpoint is taken and when Timer_pmt expires it converts tentative checkpoint into permanent checkpoint. Timer_ckp is set to T and Timer_pmt is set to T+D+2nρT, where n is checkpoint sequence number. The messages sent, in MD + tdmax time units before Timer_ckp expires, are logged by saving in queue_in_transit to avoid inconsistencies and the messages sent, in ED time after Timer_ckp expires, are sent with checkpoint sequence number (CSN), so that receiver compares and takes forced checkpoint decision depending on its CSN to avoid inconsistencies.

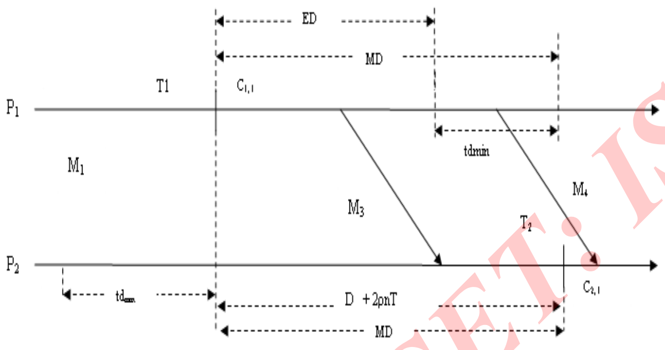


Fig. 1 Orphan Message

III. SYSTEM MODEL

Nuno Neves[5] algorithm the processes are non-blocking because the consistency problem was solved by the information piggybacked in each message. But when the transmission delay between two mobile hosts becomes relatively large, the synchronization result of processes will be less accurate. The unique feature of the scheme is use of time to synchronize the checkpoint creation. They used time to indirectly coordinate the creation of recoverable consistent checkpoints. It requires checkpoints be sent back only to home agents which results in high failure free overhead during checkpointing. We use system model as in [5].

IV. PROPOSED ADAPTIVE PROTOCOL

In this section single phase non-blocking synchronous algorithm suitable for mobile computing environments. The main advantage of the algorithm is to produce a consistent set of checkpoints without the overhead of taking temporary checkpoints. The algorithm requires only minimum number of processes to take checkpoints in any execution of the checkpointing algorithm. Performance analysis shows that our proposed algorithm outperforms some existing important related works.

Checkpoint Initiation Assumption:

Any process may become checkpoint initiator. In our approach the node only desires to send checkpoint timer to those processes from which it receives computation message(s) i.e. dependent processes. If a process Pi needs to take a checkpoint then any of the following events occurs:

If Pi is the initiator.

a) If it receives a primary checkpoint timer from the initiator.

b) The first time it receives a secondary checkpoint timer and prior to that it has not received any primary checkpoint timer or any piggyback application message.

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 timer message

Checkpoint creation procedure:

The checkpoint creation procedure can be implemented using the code. The procedure consists of two timers at which computation messages are not allowed to send. One timer expires (D+ 2nρT + tdmax) seconds before the checkpoint and another expires at checkpoint time. The process need to block only synchronous message. Computation message can store in queue. For this purpose first time call the stopSMsg function. This function save the queued messages and reset the timer. The function createchkp () is executed when the second timer expires It saves the process state, increments the checkpoint time with the checkpoint period T, and resets the timer. Next, createchkp() tests if ED seconds have passed since the checkpoint time if the condition is not satisfied, this means that the term 2ρkT has grown too large, and that timers need to be re-synchronized.

A. Notation and Data Structure:

The following notations and data structure are used in our algorithm

sndi: A Boolean array of size n maintained by each process Pi. The array is initialized to all zeroes each time a checkpoint at that process is taken. When pi sends a computation message to process Pj, Pi sets sndi[j] to one.

init: A tuple (Pid, inum) maintained by each process Pi . Where pid indicates the checkpointing initiator that initiate this process to take its latest checkpoint. Where inum indicates the csn at process Pid when it took its local checkpoint on initiating the checkpointing process. Where init is appended to each system message and the first computation message to which a process sends after taking its local checkpoint.

csni: An array of n checkpoint sequence numbers (csn) at each process Pi. Each checkpoint sequence number is represented by an integer. csni[j] represents the checkpoint sequence number of Pj that Pi knows. In other words, Pi expects to receive a message from Pj with checkpoint sequence number csni[j]. Note: csni[j] is the checkpoint sequence number of Pi

DVi[j]: An N x N dependency vector where each process show the dependency on another process which is used to built the dependency matrix D by MSSinit.

cellk: The wireless cell in which no. of processes exists are served by MSSk

Recvi: An array of N bits of process Pi maintained by Pi's local MSS. In the beginning of every checkpoint interval. Recvi[j] is initialized to zero for j = 1 to N except that Recvi[i] always equals 1. When Pi receives a message m from Pj, and the receipt of m is confirmed by Pi's MSS. Recvi[j] is set to 1.

chkp_timer: A Timer whose value is send with computation message to set the timer of receiver to take checkpoint. Local time of MSS's Timer i.e. next checkpoint time is used to set the value.

stopSnaMsg: A flag with True or False. Initially False for all processes it change to True when blocking time interval starts.

recv_csn: The recvi vector of the preceding checkpoint interval of process Pi which is maintained by Pi's local MSS.

tdmin: A timer to store the minimum message delivery time.

tdmax: A timer to store the maximum message delivery time.

pchkpt_ti: A timer whose value show the direct dependency between initiator and other processes.. To identify the primary timer initiator sends its own identity init (Pid, inum) with computation message.

schkpt_timer: A timer whose value show the transitive dependency on initiator and direct dependent on other processes to which the process send at least one message.

B. The protocol

1. *Action taken when Pi send a computation message to Pj*

```
if (sndi[j] = =0)
    {
    sndi[j] =1;
     send (Pi, msg, csni, chkpt_timer, init);
    }
    else
    {
    send (Pi, msg, csni , chkpt_timer, null);
    }
```

2. *Action for the initiator pj*

Take a local checkpoint;

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

```
csni ← csni+1;                  // Increment Checkpoint sequence number
chkp_timer = getTime () + T  // Set first checkpoint time using getTime () & checkpt period T
time = getTime ();               // Initialize time using MSS local time
setTimer (createNewChkp, Chkp_timer);          // set timer based on createchkp ( )
Intrvl = chkpTimer - MD;                        // Initiator set the interval to synchronize process
send chkpt_request message (msg, csni, chkpt_timer, init,intervl) to local MSS;  // MD = maximum
//deviation (D + 2Tρ + tdmax)
```

3. *Action at Local MSS of initiator (MSSinit):*

```
MSSckpt_time ←        T;             //  set MSSckpt_timer  T;
Check the dependency vector DVj [];
if (DVj [k] = = 1)
{
 for (i=1; i<= n; i++)
{
send (msg, chkpt_timer, intervl, csnj, init // Send message to other MSSs or processes
 }
 increment checkpoint;      // When local timer expire Take a checkpoint
}
```

4. *Action executed at MSSk*

```
recv (msg, csni, chkpt_timer, intervl, init);   // Receive message from initiator MSS
if (recv_csn<=csni[j])                        // Comparison with received checkpoint sequence number
{
 send the message to process;
   exit ();
}
else
{
 csni [j] ←   recv_csn; //Set received checkpoint sequence number as current sequence no.
 }
upon receiving message from MSSinit:
for each i such that Pi Є cellk
receiveMsg (Pj, recv_csn, chkp_timerj, intervl msg);
if ((csni ==  recv_csn) &&(chkp_timer > chkp_timerj))  // Comparison with received checkpoint
{                                                          // csn and timer
  resetTimer (chkp_timerj);
send recvi to MSSinit;            //Acknowledgement sent back to MSS initiator
}
else if (csni < recv_csn)     // Orphan message condition
{
csni  ←   recv_csn;      //Set received checkpoint sequence number as current sequence no.
 recvi[j] ←  1;
 resetTimer (chkp_timerj);
send recvi to MSSinit;       //  Acknowledgement sent back to MSS initiator
}
```

5. *Action at process Pi   when timeout event is triggered for chkpt interval:*

```
Createchkpt ();      // Take checkpoint when local timer get expires
If (DVi [] = = 1)   //Pi finds its own dependent vector
{
```

```
Send (msg, schkpt_timer, csni)    // Send secondary checkpt timer and seq. no. with msg.
}
```

6. *Any process Pk*

```
if (Pk receives a  msg with pchkpt_timer from initiator)
{
chkp_timer = time + interval - tdmin ;        // Set checkpoint timer using checkpoint interval
setTimer (createchkp, chkp_timer)            // Initiate timers using checkpoint timer
 createchkpt ( );
if (DVk [] = = 0)    // Check the DV to send the checkpoint timer to dependent processes
       {
increment csni;
process the message;
continue normal operation;
}
else
{
 send (msg, chkp_timer, csnk);
 increment csni;
process the message;
continue normal operation;
}
}
else if (Pk receives a msg with schkpt_timer )
{
if (Pk has already participated in primary checkpoint) //   Processes already participated in
{                                   // checkpointing algorithm will not take secondary checkpoint
 process the message;
continue normal operation;
}
 else
 {
schkp_timer = time + chkp_timer - tdmin;      // Computation of secondary checkpoint timer
setTimer (createchkp, schkp_timer);          // and accordingly set two timers
createchkp ();
if (DVk[ ] = = 0 )
{
increment csni;
 process the message;
continue normal operation;
 }
else
{
schkp_timer = time + chkp_timer - tdmin;
setTimer (createchkp, schkp_timer);
send (msg, schkpt_timer, csnk);
increment csni;
process the message;
operation;
}
}
```

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

```
}
```

7. *Procedure createchkp ( )*

```
SaveProcessState ( );
k = k + 1;
chkp_timer = chkp_timer + T;
setTimer (createChkp ( ), chkp_timer);
if ((D+2ρ (k-1) T – tdmin) > (getTime () - (chkp_time-T)))       // Call resy  procedure
{
RequestResyncTimers ();              // When blocking time greater than checkpoint interval
SendqueuedMessages ();                                                    // store in queue.
```

C. *Working Example*

Consider the distributed system as shown in Fig. 2 Assume that process P2 initiates the checkpoint algorithm. First process P2 takes its permanent checkpoint C2,1 when the

timer expires. P2 set the timer for other processes and define a time interval before the checkpoint creation time during which processes are not allowed to send messages. The extent of the interval is proportional to the maximum message delivery time.P2 send the checkpoint request to its own MSS. MSS set its checkpoint timer and then check its dependency vector DV2[ ] which is {1,0,1,1,0,0,0}. This means that P2 has received at least one message from processes P1,P3,P4 and P2 has already taken its checkpoint C2,1 these messages can become orphan if P1,P3,P4 do not take checkpoints. Therefore P2 send interval to P1, P3, P4 to set their timer and when the time expires take checkpoint.

Eventually processes P7 also receive the secondary timer from process P5. P7 First compare its current checkpoint sequence number with received checkpoint sequence number which is also 2 It finds that its current checkpoint sequence number is equal to the received checkpoint sequence number. Hence P7 discards it as it already takes its checkpoint for the current execution of the algorithm. In the example if there was no such piggyback message sent by process P4 then P7 would receive the checkpoint timer and set its own checkpoint timer and take checkpoint when its timer expires. Observe that proposed algorithm is nonblocking. Consider the following situation where suppose that no message was sent by process P7 to any process at all. However assume that it receives the piggyback message from P4 and take checkpoint then it process the message. This represent the consistent state of process P7 This means that process P7 would start resynchronization from C7,1 rather than C7,0 after system recovery from failure.

D. Proof of correctness

Theorm 1: Proposed Algorithm Non- blocking produces a consistent global state of the system.

Proof: Firstly, the initiator process pi identifies all the application messages received from different processes that might become orphan if it takes a checkpoint by looking at its dependency vector. (a) The initiator then sends primary checkpoint timer with computation message to all dependent processes. On time expire all dependent processes takes their respective checkpoints. Hence any application message received by pi cannot be an orphan and also the process make sure from which it receives messages also take checkpoint so that there are no orphan messages that it has received (b) As the timer are not well synchronized. The process can piggyback the checkpoint number with computation message. Hence such a message cannot be an orphan. Hence the algorithm generates a consistent global state of the system.

Theorm 2: Number of processes take checkpoint is minimum.

Proof: A process takes checkpoint if and only if it is the initiator, or it receives either a primary checkpoint timer or secondary checkpoint timer or a piggyback application message. This means that except these condition other processes does not take a checkpoint. Hence, the proof.

Theorm 3: Avalanche Effect does not occur in proposed approach.

Proof: Consider the Following situation: suppose process pi initiate the synchronous time based checkpointing scheme. It takes a checkpoint. Check the dependency vector and send the primary checkpoint timer to dependent processes. Suppose pj receives the primary

Vol. 2 Issue IV, April 2014
ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

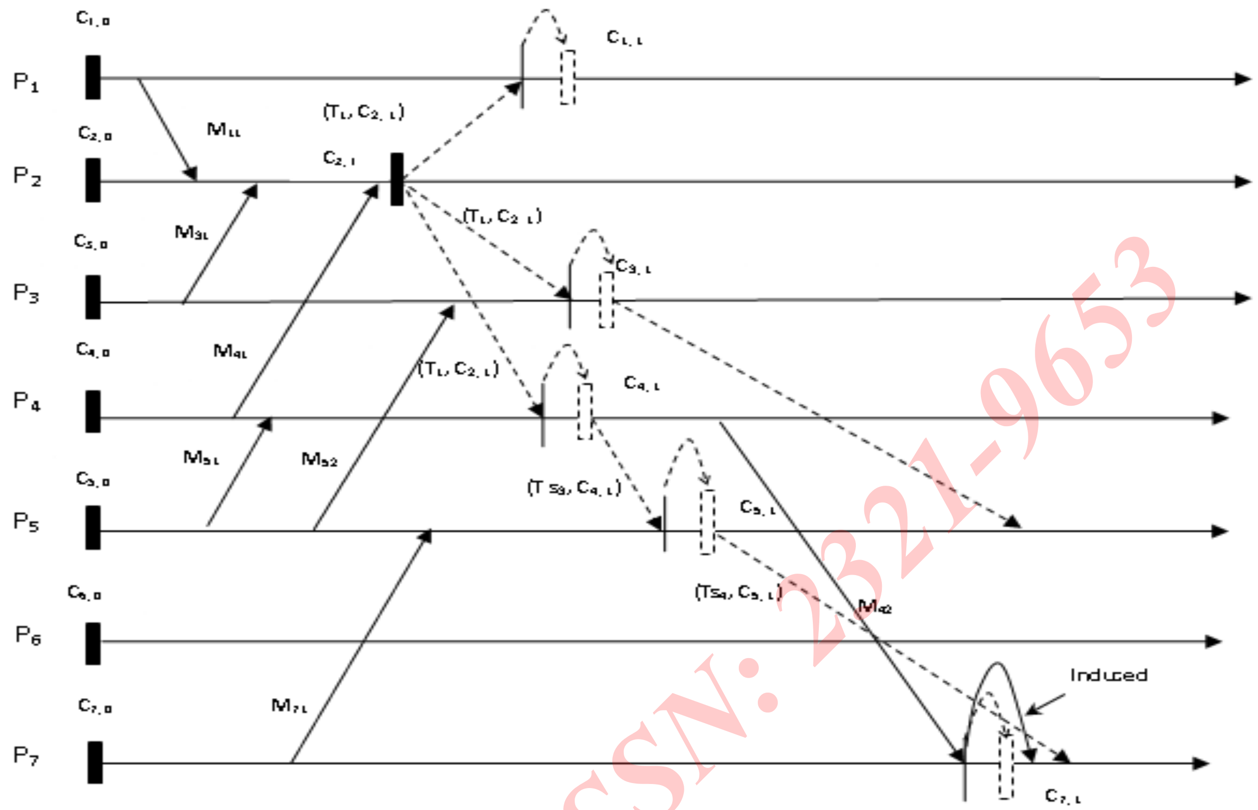


Fig. Working example of proposed scheme

V. PERFORMANCE ANALYSIS

The performance analysis of proposed algorithms is made on the basis of blocking time, synchronization message overhead, number of processes required to checkpoint, piggybacked information on to computation messages Table 1. Detailed performance analysis is explained in [13.] Let's the following notations are used:

N_{mss}: Number of MSSs.
N_{mh}: Number of MHs.
N_{min}: Number of minimum processes required to take checkpoints.
N_{mut}: Number of useless mutable checkpoints.
N_{ind}: Number of useless induced checkpoints.
N: Total number of Processes.
C_{wired}: Cost of sending message from MH to its local MSS (or vice versa).
$C_{wireless}$: The cost of sending message between processes.
N_{broad}: The cost of broadcasting a message to all processes in the system.

T_{ch}: Average delay to save a checkpoint on the stable storage. It also includes the time to transfer the checkpoint from an MH to its local MSS.

TABLE 1 Performance and comparisons of Rollback recovery Algorithms

Algorithm/ Parameter	**Mutable[1]**	**Non Intrusive [2]**	**CCUML [3]**	**Neves-Fuches [7]**	**C.Lin, Szu chi [10]**	**Proposed protocol**
Cost/Overhead	$2{*}N_{min}{*}C_{wireless} + \min(N_{min}{*}C_{wireless}, N_{broad})$	$N{*}C_{wireless} + 2{*}N_{min}{*}C_{wireless} + 2{*}N_{broad}$	$N{*}C_{wireless} + 2{*}N_{broad}$	$\sigma + 2\rho_{MH}Ttd_{min}$	$(N-N_{min}) \times (C_{wired} + C_{wireless}) + N \times C_{wired}$	$N_{min}{*}C_{wireless}$
Useless checkpoint	Present	Nil	Nil	Yes	Yes	Nil
Non-Blocking	Yes	Yes	Yes	Yes	Yes	Yes
Number of checkpoints	N_{min}	N_{min}	N	N	N_{min}	N_{min}
Output Commit	$N_{min}{*}T_{ch}$	$N_{min}{*}T_{ch}$	$N{*}T_{ch}$	$N{*}T_{ch}$	$N_{min}{*}T_{ch}$	$N_{min}{*}T_{ch}$

VI. CONCLUSION

In the proposed protocol first the initiator sends the control messages to minimum number of processes that need to take a checkpoint each. The cost for this is Nmin* Cwireled. The protocol uses time to indirectly synchronise the processes, and ensure the consistent and recoverability during fault. No control messages are used between processes. Some control messages are being sent between processes to local MSS. It definitely offers much better bandwidth utilization and face minimum number of interrupt. Frequently changing the network of MHs does not affect to the protocol as it takes soft as well as hard checkpoints to adapt the behaviour of system. Soft checkpoint are used in most cases to reduce overheads and after sometimes hard checkpoints taken to guarantee that permanent failures can be tolerated. The number of checkpoints used in the algorithm is balanced and the number of useless checkpoint is nil which offer better utilization of the mobile host's limited memory.

REFERENCES

[1] Guohong Cao and Mukesh Singhal, "Mutable Checkpoints: a new checkpointing approach for Mobile Computing Systems", IEEE Transaction on Parallel and

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Distributed Systems, vol. 12, no. 2, pp. 157-172, February 2001.

[2] Parveen Kumar, Lalit Kumar, R K Chauhan, V K Gupta, "A non-intrusive minimum process synchronous checkpointing protocol for mobile distributed systems", Proceedings of IEEE ICPWC-2005, IEEE International Conference on Personal Wireless Communications, pp 491-495, January 2005, New Delhi.

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

[4] J.L.Kim and T.Park. "An efficient protocol for checkpointing recovery in Distributed Systems" IEEE Transaction on Parallel and Distributed Systems, 4(8): pp. 955-960, Aug 1993.

[5] Nuno Neves and W. Kent Fuchs. "Adaptive Recovery for Mobile Environments", in proceeding IEEE High-Assurance Systems Engineering Workshop, October 21-22, 1996, pp.134-141.

[6] C. Lin, S. Wang, and S. Kuo, "An efficient time-based checkpointing protocol for mobile computing systems over wide area networks," in Lecture Notes in Computer Science 2400, Euro-Par 2002, Springer-Verlag, 2002, pp. 978–982. Also in Mobile Networks and Applications, 2003, vo. 8, no. 6, pp. 687–697.

[7] N.Neves, W.K.Fuchs, "Using time to improve the performance of coordinated checkpointing," In: Proceedings of 2nd IEEE International Computer Performance and Dependability Symposium, Urbana-Champaign, USA, 1996, pp.282 –291.

[8] D.Manivannan and M.Singhal, "A low-overhead recovery technique using quasi- synchronous checkpointing," Proc. 16th Int. Conf. on Distributed Computing System, 1996, pp.100 -107.

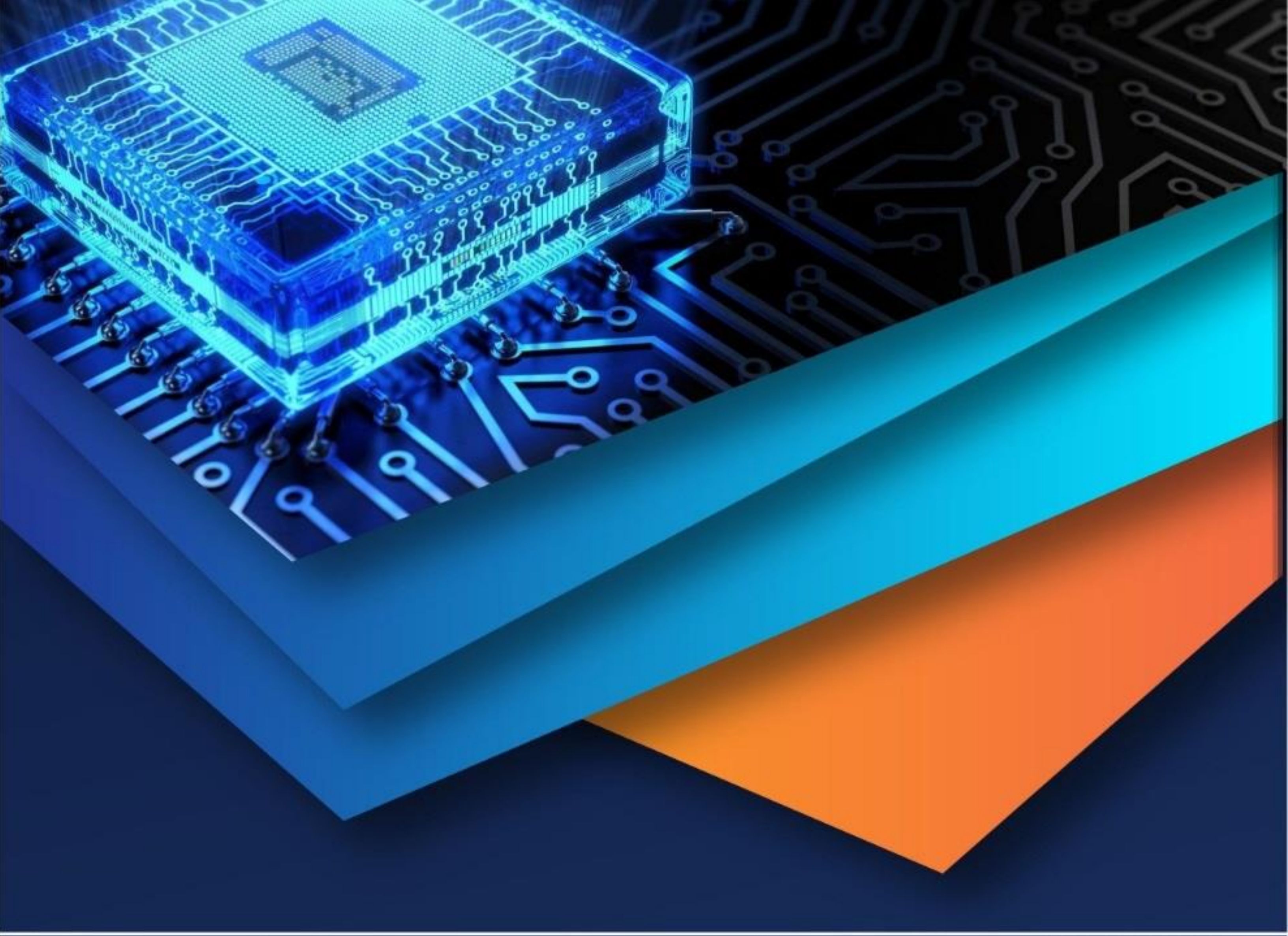
[9] M. Chaoguang, Z. Yunlong, and Y. Wenbin, "A two-phase time-based consistent checkpointing strategy," in Proc. ITNG'06 3rd IEEE International Conference on Information Technology: New Generations, April 10-12, 2006, pp. 518–523.

[10] C. Lin, S. Wang, and S. Kuo, "A Low Overhead Checkpointing Protocol for Mobile Computing System" in Proc of the 2002 IEEE Pacific Rim International Symposium on dependable computing (PRDC'02).

[11] Sourav Basu, S. Palchaudhuri, S. Podder, M. Chakrabarty", A Checkpointing and Recovery Algorithm Based on Location Distance, Handoff and Stationary Checkpoints for Mobile Computing Systems", International Conference on Advances in Recent Technologies in Communication and Computing pp 58-62,27-28 October 2009 .

[12] Jangra Surender, Sejwal Arvind, Kumar Anil, Sangwan Yashwant "Low Overhead Time Coordinated Checkpointing Algorithm for Mobile Distributed Systems", Published by Springer in Lecture Notes in Electrical Engineering ,Volume 131 , Page no. 173-182.

[13] Surender Kumar, R.K.Chauhan and Parveen Kumar, "Designing and Performance Analysis of Coordinated Checkpointing Algorithms for Mobile Distributed Systems", International Journal of Distributed and Parallel Systems [IJDPS] (AIRCC France), Vol.1, No.1, pp. 61-80, Sept. 2010.



10.22214/IJRASET

INDEX COPERNICUS 45.98

ISRA JIF IMPACT FACTOR: 7.129

SJIFactor TOGETHER WE REACH THE GOAL IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY