iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2 **Issue:** V **Month of publication:** May 2014

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

Vol. 2 Issue V, May 2014
ISSN: 2321-9653

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

Soft-checkpoint based Checkpointing Algorithm for Mobile Distributed Systems

Pradeep Kumar Sharma[1], Parveen Kumar[2], Surender Jangra[3]

[1]Research Scholar, Deptt. of Computer Sci., Mewar University, Chittorgarh(Raj.)
[2] Deptt. of CSE, Bharat Institute of Engg. & Tech. Meerut(UP)
[3] Deptt. of IT Engg. HCTM Technical Campus, Kaithal(HRY)

Abstract-- There are number of issues needs to handle in mobile computing systems like mobility, lack of stable storage on mobile nodes, disconnections, limited battery power and high failure rate of mobile nodes which causes loss of computation. Checkpointing is an attractive approach to introduce fault tolerance in mobile distributed systems transparently. A checkpoint is a local state of a process saved on the stable storage. However, a mobile consumer device is not considered to have sufficiently large and stable storage to store its checkpoint data. Therefore, a remote checkpoint technique is preferred in which the checkpoint data of a mobile device is kept in a remote checkpoint server instead of the mobile device. "Soft checkpoint" is neither a tentative checkpoint nor a permanent checkpoint, which can be saved anywhere, e.g., the main memory or local disk of MHs. Before failure these soft checkpointsare converted to hard checkpoints and are sent to MSSs stable storage. In this way, taking a soft-checkpoint avoids the overhead of transferring large amounts of data to the stable storage atMSSs over the wireless network and minimizes power consumption.

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

I. INTRODUCTION

The market of mobile handheld devices and mobile application is growing rapidly. Mobile terminal are become more capable of running rather complex application due to the rapid process of hardware and telecommunication technology. Property, such as portability and ability to connect to network in different places, made mobile computing possible. Mobile computing is the performance of computing tasks whiles the user in on the move, or visiting place other than their usual environment. In the case of mobile computing a user who is away from his "home" environment can still get access to different resources that are too computing or data intensive to reside on the mobile terminal [4]. Mobile distributed systems are based on wireless networks that are known to suffer from low bandwidth, low reliability, and unexpected disconnection [4].

In MDS multiple MHs are connected with their local MSS through the wireless links. A process is considering as a MH without stable storage. During checkpointing, an MH has to transfer a large amount of checkpointed data (like variables, control information, register value and environments etc.) to its local MSS over the wireless network. So, checkpointing consumes scares resources to transfer data store data on stable storage. If even a single MH fails to take a checkpoint, all the checkpoints which are taken related to current initiation must be discarded. So, it increases the overhead.

Mostly coordinated checkpointing approach follows two-phase commit structure. In the first phase, processes take tentative checkpoints and in the second phase, these tentative checkpoints are made permanent. If a single process fails to take its checkpoint or reply negatively, the whole checkpointing effort of the particular initiation become wasted and processes rollback to previous CGS. Checkpointing takes some time to operate as all the computed data and related information are transferred and stored on stable storage which takes some time to save, transferring and also consume power.

In mobile distributed systems (MDSs) processes are considering as many MHs with low power, memory and without stable storage. There is wireless link between MH and MSS. So during checkpointing in MDSs, an MH has to transfer a large amount of checkpoint data to its local MSS over the wireless network. Since MHs are prone to failure and a single MH failure made the checkpoints efforts unsuccessful. After a single failure all the related processes rollback to its previous consistent state. Therefore, such unsuccessful efforts consume a lot of time, waste scarce resources of mobile systems likes' computation power, bandwidth, time, energy and it is performance bottleneck.

A. Why Soft-checkpointing Approach

In this section we compare our soft checkpoint (SC) based approach with the hard Checkpointing (HC) approach in different perspectives.

Checkpoint Latency: Checkpointing latency is the time needed to save the checkpoint. It is observed that there is a big difference between the latency in soft and hard checkpointing based approach. This is due to fact that soft checkpoint based approach uses fast volatile memory and not have any transmission cost.

*Transmission Cost***:** Soft checkpoint based approach have not any transmission cost as the checkpoint are stored locally on volatile memory, instead through wireless link on stable storage of MSS.

Recovery Time: It is observed that recovery time in soft checkpoint based approach is much less if the failure occurs before converting the SC into HC and it is comparable in the other case.

B. Soft vs Hard Checkpointing

As shown in Table 1, soft-checkpoint based approach has low latency, cost and overheads compared to hard approach but it is assumed that soft checkpoint approach is very less reliable. In our approach to increase the reliability of approach, on the time of failure, disconnections or handoffs we transfer the data on the stable storage of MSS.

TABLE 1 HARD VS SOFT CHECKPOINTING

Parameter	Hard-based	Soft-based
Checkpoint Latency	High	Low
Transmission Cost	High	Low
Recovery Time	High	Low
CPU Overhead	High	High
Main Memory Requirement	Low	High
Reliability	*High*	*Low*
Efficiency	Low	High
Portability	High	Low
Additional Hardware	Not required	Additional processors
Suitability	For Large Systems	For Small Systems
Power Consumption	High	Low

II. RELATED WORK AND PROBLEM FORMULATION

A. Related Work

Most of the coordinated checkpointing protocols [4], [6] take forced or mutable checkpoints to achieve the goal of non-blocking and minimum process. Many of these checkpoints may not become the part of CGS and such types of checkpoints are treated as useless. Useless checkpoints are not desirable because they do not contribute to the recovery of the system from failure but they consume resources and increase the performance overheads.

Major objective in the designing of a checkpointing protocol is to make it non-blocking and forces a minimal number of processes to take their local checkpoints and reduces the rollback overhead.

Cao and Singhal [6] achieved non-intrusiveness in the minimum-process protocol by introducing the concept of mutable checkpoints. If any process sends a computation message to another process after receiving the checkpoint request, the receiving process first take the mutable checkpoint first and process the message. Later, this mutable checkpoint converted to tentative if it receives checkpoint request related to the current initiation; otherwise it become the useless checkpoint. Besides as in [4] and [6], our checkpointing algorithm requires minimum checkpoints and reduces useless checkpoints.

The most commonly used technique to prevent complete loss of computation upon failure is Coordinated checkpointing [2], [4], [5], [6], [13]. In this approach, the state of each process in the system is periodically saved on the stable storage, which is called a checkpoint of the process. To recover from a failure, the system restarts its execution from a previous consistent global checkpoint saved on the stable storage. In order to record a consistent global checkpoint, processes must synchronize their checkpointing activities. In other words, when a process takes a checkpoint, it asks to all relevant processes by sending checkpoint requests to take checkpoints. Therefore, coordinated checkpointing suffers from high overhead. The protocol presented in this paper shows performance improvement over the work reported in [4], [5], [6], [7] & [8]. The protocol designed by Acharya and

Badrinath [1] requires to create a new checkpoint whenever they receive a message after sending a message. Processes also have to create a checkpoint prior to disconnection. Pardhan et al. [7] proposed two uncoordinated protocols. The first protocol creates a checkpoint everytime when a process receives a message. The second protocol creates checkpoints periodically and logs all messages received. P. Kumar and R. Garg [11] proposed a hybrid scheme, wherein an all process checkpoint is enforced after executing minimum-process algorithm for a fixed number of times. In the first phase, the MHs in the minimum set are required to take soft checkpoint only. Soft Checkpoint proposed by them is stored on the disk of the MH and is similar to mutable checkpoint [8]. In the minimum process algorithm, a process takes its forced checkpoint only if it is having a good probability of getting the checkpoint request; otherwise, it buffers the received messages.

S. Kumar et. all [12] proposed a soft checkpoint approach in which aprocess in minset [ ] takes a soft checkpoint first andthen soft checkpoint will be discarded, if it receives aborted message from the initiator. These soft checkpoints are saved on main memory of the mobile hosts [MHs], and then the soft checkpoint will be saved on the stable storage of MSS at a later time only if they receive the hard checkpoint request from the initiator. Their scheme requires low battery power of MHs, low checkpoint latency, low transmission cost, and low recovery time due to reduced disk accessed of MSS by the MHs. As soft checkpoint approach is less reliable, to make it reliable they transfer the soft checkpoint on stable storage The protocols proposed in [4], [5] & [8] follow two-phase commit distributed structure. In the first phase processes take temporary checkpoints when they receive the checkpoint request. These tentative checkpoints are stored in stable storage of MSS. In the second phase, if an MSS learns that all the processes have taken the temporary checkpoints successfully, initiator MSS sends commit message to all the participating nodes. In these checkpoints an MH has to transfer a large amount of data to its local MSS over its wireless network which results in higher checkpoint latency and recovery time as transferring such temporary checkpoints on stable storage may waste a large amount of computation power, bandwidth, energy and time. The protocol proposed by us creates a checkpoint whenever the local timer expires, and it only logs the unacknowledged messages at checkpoint time. Our protocol uses two types of checkpoints to recover from failure. The two previous protocols proposed in [6] and [7] always assume hard failures.

B. Problem Formulation

The objective of the present work is to design a checkpointing approach that is suitable for mobile computing environment. Checkpointing and recovery protocol for mobile environment demands for efficient use of the limited resources of mobile environment i.e. wireless bandwidth, battery power and memory etc. Consider a mobile environment, in which a MSS has 1000 MHs in their minimum set. During the first phase 999 MHs take their temporary checkpoints successfully and one MH fails to take checkpoint. In such case, the checkpointing process must be aborted and MHs discard their temporary checkpoint. Observe that taking a temporary checkpoint in the stable storage and later discard it, affects the bandwidth utilization and waste the MHs limited battery power.

III. Proposed Soft-Checkpoint based Algorithm

A. Architecture

Our checkpointing algorithm has two types of checkpoints: permanent and soft checkpoint. Permanent checkpoints are same as in [4], [5], [6] and saved on stable storage of MSS. Soft checkpoint are like forced checkpoint and need not to be saved on stable storage and can be stored anywhere, even in the main memory of MHs. This approach assumes that each site has its own local log, sufficient volatile memory and can therefore rollback or commit the transaction reliably. This soft checkpoints based approach [Fig. 1] is suitable for MDSs as in case of single MH failure processes rollback to previous consistent state without incurring extra cost as soft checkpoints are stored on local MHs and have not any transferring cost.

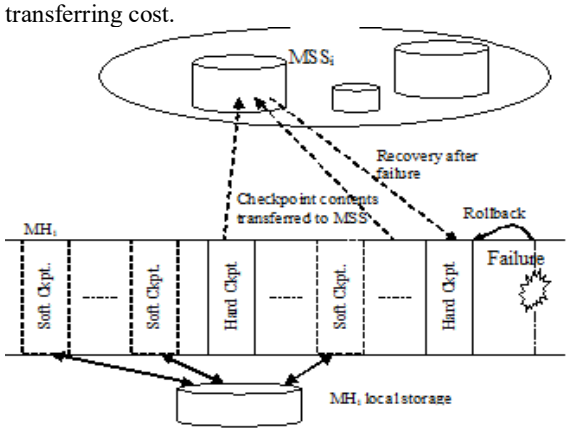


Fig. 1 Soft-checkpoint based architecture

These soft checkpoints are transferred to stable disk of MSS only after receiving the *commit*. On receiving the abort message from the initiator or identifying the participated node failure these soft checkpoints are discarded locally. Hence,

during failure our soft checkpoint has much less overhead as compare to tentative.

B. Basic Idea

When an initiator process P_i, initiate the checkpointing process, it takes its local checkpoint and sends soft checkpoint request piggybacked with the tuple (Pid,icsn) of initiator to all directly and transitively dependent processes to take their soft checkpoints; and wait for response. After receiving the checkpoint request from the initiator process it depends upon the processes they take their soft checkpoint or not. If one process fails to respond within a timeout period, then the initiator broadcast ABORT message. After receiving *ABORT*, processes discard their checkpoints taken related to current CI. On the other hand if initiator knows that all processes take their soft checkpoint successfully, then it forwards *COMMIT* message to all related processes and processes converts their soft checkpoints into permanent checkpoints after receiving the *COMMIT* message.

C. System Model

A MDS is a distributed system where some of the processes are running on mobile hosts (MHs) [3]. It consists of Static Hosts (SHs), Mobile Hosts (MHs) and the Mobile Support Stations (MSSs). So, the MDS can be considered as consisting of "n" MHs and "m" MSSs. The static network provides reliable, sequenced delivery of messages between any two MSSs, with arbitrary message latency. Similarly, the wireless network within a cell ensures FIFO delivery of messages between an MSS and a local MH. The links are FIFO in nature. An MH communicates with other nodes of system via special nodes called mobile support station (MSS).An MH can directly communicate with an MSS only if the MH is physically located within the cell serviced by MSS. A static node that has no support to MH can be considered as an MSS with no MH. A cell is a geographical area around an MSS in which it can support an MH .An MH can change its geographical position freely from one cell to another cell or even area covered by no cell. At any given instant of time an MH may logically belong to only one cell; its current cell defines the MH's location and the MH is considered local to MSS providing wireless coverage in the cell. If an MH does not leave the cell, then every message sent to it from local MSS would receive in sequence in which they are sent.

D. Data Structure

Each process P_i maintains the following data structures:

Pid	Initiator process identification;
M	m is any process to process computation message;
$csn_i[]$	An array of checkpoint sequence numbers maintained by each process P_i. It is initially set to zero and increı takes a new checkpoint by one.
$csn_i[j]$	Checkpoint sequence number (csn) of P_j currently known by the process P_i.
$weight_i$	Weight is used to detect the termination of the checkpoint algorithm as in [4], [5], [6]. Initiator process sends the portion of weight with the checkpoint request and receiving process reply with weight to the initiator. When initiator process found weight =1, it sends the commit () message to all processes belongs to the minimum set
$trigger_i$	A set of 2-tuple (Pid, icsn) maintained by each process P_i, where P_{id} indicates the initiator process identifier of the checkpointing initiator and icsn indicates the csn at process P_{id} corresponding to that checkpoint event.
$ddv_i[]$	A bit vector of size n; $ddv_i[j]$ =1 implies P_i is directly dependent upon P_j for the current CI; $ddv_i[j]$ is set to '1' only if P_i processes m received from P_j s.t. m.own_csn □ $csn_i[j]$; otherwise $ddv_i[j]$=0. m.own_csn is the own_csn at P_j at the time of sending m and $csn_i[j]$ is P_j's recent permanent checkpoint's csn; initially, □k, $ddv_i[k]$=0 and $ddv_i[i]$=1; for MH_i it is kept at local MSS;
c_state	A flag set to "1" when a process P_i takes soft checkpoint after receiving the checkpoint request.

minset$_i$[]	In order to maintain minimal dependency information among processes we use the similar approach as [6] where Boolean vector R_i of n bits. $ddv_i[i]$ =1 or $ddv_i[j]$, if j≠1. A process P_i sends an execution message m piggybacked with ddv_i to P_j. P_j updates its own ddv_j after receiving m as follows: $ddv_j[k]$= 1 or $ddv_i[k]$ m.ddv[k], 1≤k≤n and v is the bitwise inclusive OR operator. Hence if P_i depends on P_k before sending m, P_j also become dependent transitively on P_k after receiving m. A bit vector of size n which is compute on the MSS_{ini}; if Pi initiate its(x+1)th checkpoint then the set of processes on which Pi depends(directly or transitively) in its x^{th} checkpoint interval is minimum set. MSS_{ini} computes minset[](subset of minimum set) on the basic of ddv[] maintained at MSS_{ini}. Initially minset[]= ddv_{ini}[]. So minset[k] =1 implies P_k belongs to the minimum set and it is directly or transitively dependent on initiator process P_{ini} .
Uminset$_i$	It is an updated version of minset at initiator process P_i. As our process is non-blocking and dependent process direct acknowledge to the initiator process. If a new dependency occurs due to non-blocking nature and this new process P_k directly reply to the initiator process then initiator process adds this new process in to minset. So now Uminset=minset U P_k.
SC	Each process P_i takes soft checkpoint when it receives the checkpoint request from the initiator or if receives the triggering computation message and $icsn_i[j] < m.csn_j$.
PC	Permanent checkpoint (PC) is stored on the stable disk of the local MSS. Our soft checkpoint is converted into permanent on receiving the commit request, during node failure and disconnections.

E. The Proposed Algorithm

(a) *Action on the initiator P_j:* Send checkpoint request to the local MSS.

(b) *Action at the MSS_{ini}*

1. if g_chkpt= =0
   {set g_chkpt= = 1;
   set c_state= =1; csn_j++;
   set init_trigger(pid,icsn);
   check $minset_j$[] ;
   Send soft chk_req to all the processes in minset i.e $minset_i[k]$ = =1 for i<=k<=n with ($minset_j$, NULL, P_j, init_trigger, weight);
   }
   else
   {Ignore request; as some global chkpting initiation is already going on ;}
2. Wait for reply ;
3. On receiving Reply from any process P_i
   Receive c_rply(P_i, reply, recv-weight)
   if (timeout)OR (Negative ACK)
   {Broadcast ABORT and exit ;}
   else
   {weight: =weight + recv_weight;
   Uminset: = minset U P_j }
4. if weight=1
   Sends commit message to all process P_k such that P_k belongs to Uminset[];

(c) *Action taken when any P_i sends a computation message to process P_j*

if SC= =T
{Send(Pi, msg, Ri, csn_i[i], trigger);
else
Send (Pi,msg, Ri, csn_i[i], NULL);}

(d) *Action taken when other processes, P_i receives a chkpt reqst from the initiator P_j:*

P_i receives a checkpoint request;
if SC= = F
Take checkpoint; set SC= = T, Increment csn_i; set own_trigger= = init_trigger;
if (DV_i[]= null) V DV_i[] = minset[] ;
Send c_rply(P_i, reply, recv_weight, null) to MSS_{ini};
Continue computation;
else
Sends checkpoint request to some process P_k , where k s.t. ddv_j[k]= =1 ^ minset[k]= =0) with some portion of weight and init_trigger;
Send c_reply(Pi,reply, remaining weight, new_DV[P_k]) to the initiator;
else // SC= = T // it has already participated in CI
Ignore the checkpoint request;
Continue computation;

(e) Actions for process P_i, on receiving a computation message from P_j: When process P_i receives the init_trigger with the computation message, it understands that process P_j sends the computation message after taking soft checkpoint (SC). On the other hand process only sends the message sequence number (csn) of the message.

if m.trigger != null

if (csn_i[pid] = m.init_csn) // P_i already participated in CI
{Receive message m;
Update csn_i[j];
Continue normal operation;
}
else if (csn_i[j] < m.csn_j)
{Take soft checkpoint (SC);
Receive message m;
Update csn_i[j];
Continue normal operation;
}
else
{Receive message m;
Update csn_i[j];}

(f) *Process takes the soft checkpoint on happening of any below condition first:*

- On the receipt of Take soft checkpoint request from initiator;
- Upon receiving the computation message, if (m.csn > csn_i[k]);
- If local time to take soft checkpoint expires

(g) Processes converts soft checkpoint in to permanent:

- On receipt of the commit message from the initiator;
- At the time of node disconnection and Handoff: in MDS, a MH may get disconnected voluntarily and non-voluntary. In voluntarily disconnections, MH transfer its soft checkpoint in to the stable storage of MSS before disconnection. We call this disconnected checkpoint. In such case the MSS who receive the disconnected checkpoint, coordinate on the behalf of disconnected MH. Non-voluntary disconnections are treated as fault [51].
- On the basic of maxsoft: The number of SC stored per hard HC is called maxsoft, and it depends on the quality of service of current network [9].

(h) Upon receiving COMMIT message

- Make soft checkpoint permanent;

- Increment own csn;

(i) Upon receiving ABORT message

- Discard soft checkpoint taken related to current initiation from the volatile memory;
- Roll back to previous permanent checkpoint;

F. Working Example

Consider a distributed system with n processes. P_0, P_1, P_2, P_3 and P_4. Each process P_i in the system maintains ddv_i vector of size n which is initially set to zero and an entry in $ddv_i[j]=1$ when P_i receives a commutation message during current checkpoint interval(CI). A process P_i sends a computation message m with ddv_i [] to P_j. P_j updates its own ddv_j after receiving m as follows: $ddv_j[k] = 1$ or $ddv_i[k]$ m.ddv[k], $1 \leq k \leq n$ and $\vee$ is the bitwise inclusive OR operator. Hence if P_i depends on P_k before sending m, P_j also become dependent transitively on P_k after receiving m. Also each process P_i piggybacks its icsn with only every first outgoing computation message to process P_j after taking checkpoint. Each process P_i needs to take soft checkpoint if any of the following events occurs:

1) if it receives checkpoint request from the initiator
2) if it receives a piggybacks message and with higher csn

In our proposed algorithm we assume triggering and non-triggering computation message. A computation message which are piggybacks with trigger tuple are called triggering message which is sent after taking a soft checkpoint. We explain the behavior of our algorithm with the help of following example by considering the five processes from P_1 to P_5 in a distributed system [Fig. 2]. We assume that process P_2 initiates the checkpointing process. The ddv[] vector are maintained as follows: When process P_4 sends a message m4 to P_3, it appends $ddv_4[00001]$ to the message. When P_3 receives message m4, it extracts the boolean vector ddv_4 [] from the message and updates its ddv_3 by taking the OR of ddv_3 [00010] and ddv_4 [00001].Now the updated ddv_3 become [00011] and sends this updated vector by appending with the message m2. Similarly by updating ddv_2 [] on the receipt of message m2 at process, ddv_2 [] become [00111]. In same way P_1 updates its ddv_1 [01000] to [011000] on receiving the message m1 from process P_2 and sends this updated ddv_2 [] vector with message m3. At last after receiving the message m3 from P_1 the ddv_2 become [011111] which shows that process P_1,P_3, P_4 are directly or transitively dependent on P_2. We make the observations related to our algorithm (a) Process P_1, P_3 and P_4 are the part of minimum set and receives the checkpointing request from the initiator process P_2 (b) Checkpoint $C_{0,1}$ and $C_{2,1}$ are the permanent checkpoint and stored at stable storage of MSS and non-triggering message are sent after these (c) Checkpoints $C_{4,1}$, $C_{3,1}$, and $C_{1,1}$ are the soft checkpoint, and triggering computation message are sent after these. So at the time of initiation, P_2 take its own permanent checkpoint and sends checkpoint request to P_1, P_3 and P_4, increment its csn $C_{2,0}$ to $C_{2,1}$ and wait for reply or acknowledgement.

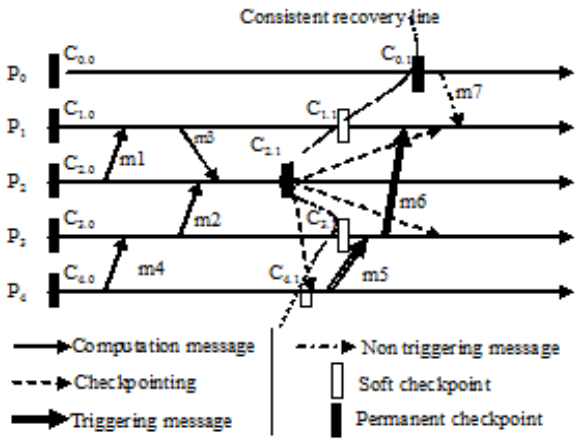


Fig. 2 An example of proposed Soft checkpointing approach

When P_2's request reaches P_4, it takes soft checkpoint $C_{4,1}$ and sends triggering computation message(as it is sent after taking the soft checkpoint) m5 to P_3. Process P_3 receives triggering computation message before it receives the checkpoint request message from the initiator, it first compares ($csn_3[P_{id}] \neq m.P_{id}$) and concludes that P_3 does not participate in current checkpointing initiation and asked to participate. Secondly it takes the soft checkpoint $C_{3,1}$ before processing the message m3 as the $csn_3[4]$ which is 1, smaller than P_4's current csn_4=2 received with message m5.

Similarly process P_1 takes its soft checkpoint $C_{1,1}$ after receiving the triggering computation message from P_3. Later when P_1 and P_3 receive the checkpoint request from the initiator P_2, these processes ignore the checkpoint request as they have already participated in current initiations. Process P_0 which are not the part of minimum set, takes its permanent checkpoint $C_{0,1}$ independently and sends the non-triggering computation message m6 to process P_1 by appending $C_{0,1}$. After receiving non-triggering computation message m6 process P_1 observe that $C_{0,1}$ is not soft checkpoint; it only

update the $csn_1[0]$ to 1, receives the message and conclude that new checkpoint is not necessary.

At last when initiator processes P_2 receives reply within time from all its dependent processes i.e., weight become 1, it sends *COMMIT* request to all the participated processes to convert their soft checkpoints into permanent one. On the other hand if time out or any one the participated process reply negatively, then initiator P_2 sends the *ABORT* message to all the participating process. After receiving the *ABORT* message, processes discard their soft checkpoint taken related to current CI.

IV. Comparative Analysis and Characteristics

a) Blocking Time: Proposed checkpointing algorithm is non-blocking.

b) Number of Coordinated Messages: In the algorithm, before failure only coordinated message soft checkpoint request are transmitted through wireless link from the MSS to MHs. The approximate number of soft checkpoints message between MSS to MHs is soft checkpoint request to $N\text{-}N_{min}$ and receives reply. Messages sent in the wired link are N receives vectors from each MSS to MSS_{ini}, and $N\text{-}N_{min}$ commit notification from MSS_{ini} to MSSs.

d) Number of useless checkpoints: Our simulation model is similar to [1]. For simplicity, concurrent initiations and failures are not considered. Each process sends out computation messages with the time interval following an exponential distribution. We plot the average number of useless checkpoints for [4], [5] and the proposed algorithm in example shown in Fig. 3.

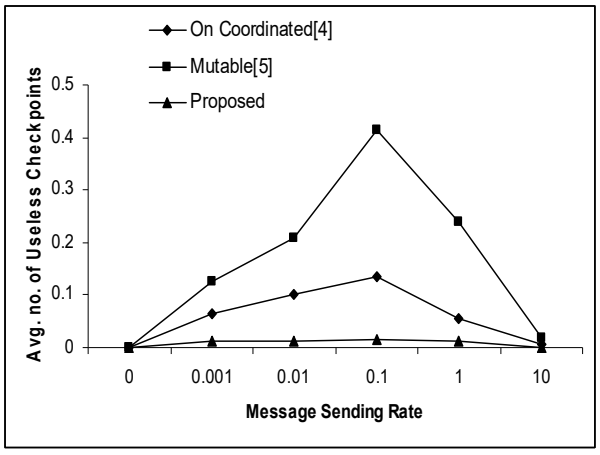


Fig. 3 Comparison of useless checkpoint

It shows that proposed algorithm takes much reduced useless checkpoints which are nearest to minimum. The number of useless in checkpoints in algorithm [5] first increases in a higher rate on increasing the message sending rate and then decreases but in algorithm [4] it first increases in a low rate and finally decreases. This is because as the message sending rate increases, more messages are sent during checkpointing, which results in increase in mutable checkpoints. As the message sending rate increases from 0.1 to 10, more and more processes are included in the minimum set; therefore, average number of a useless checkpoint keeps on decreasing.

In Cao and Singhal algorithm [5], a checkpointing tree is formed and the number of useless checkpoints may be exceedingly high in some situations. Hence, due to non-formation of checkpointing tree, the number of useless checkpoints in the proposed scheme is very small.

e) Checkpoint overhead: checkpoint overhead is the increase in execution time of application due to checkpointing. Our soft checkpoint based approach have less checkpoint overhead compared to hard as the soft checkpoint are stored on the local memory of the MHs and does not have transferred cost.

g) Recovery time: This is the time taken by a system to restore its last consistent checkpointed state after the failure. Proposed algorithm takes very less time to restore its last checkpointed state.

V. Conclusion

In this paper, soft checkpoint based non-blocking coordinated checkpointing algorithm is proposed which efficiently deal with many new issues i.e., mobility of MHs, lack of stable storage at MH's, low bandwidth of wireless channels and limited battery life of MH's. The algorithm has reduced overheads in term of number of coordination message, minimizing the number of checkpoints and useless checkpoints which is the basic requirements for MDSs. The proposed approach also provides fast recovery in case of a single node failure as soft checkpoint is stored on local disk of MHs. It is said that soft checkpoint are not reliable as during failure soft checkpointed data lost as it is stored on volatile memory. In our approach during failure or disconnections of a node these soft checkpoint are converted in to permanent checkpoint. our algorithm also reduce power consumption because (1) if an MH is in active mode, its power consumption is highest: in our approach power consumption is low as it forces only minimum numbers of processes which are directly or transitively dependent on MSS_{ini} during current CI and does note awake the processes in doze mode. (2) As

message sending consumes more power: our approach consumes less power as minimum checkpointed data is transferred through the wireless link as our soft checkpoints are stored locally on the MHs. (3) Taking extra checkpoints also consumes power: our soft checkpointing approach takes only reduced number of checkpoints nearest to minimum, so power consumption in our is low.

REFERENCES

[1] Acharya A. and Badrinath B.R., "Checkpointing Distributed Application on Mobile Computers", in the Proc. of the 3rd Int'l Conf. on Parallel and Distributed Information Systems, pp. 73-80, Sept. 1994.

[2] Koo R. and Toueg S., "*Checkpointing and Roll-Back Recovery for Distributed Systems*", IEEE Trans. on Software Engg., Vol.13, No.1, pp.23-31, Jan. 1987.

[3] Chowdhury C. and Neogy S. "*Checkpointing using Mobile Agents for Mobile Computing Systems*", Int'l Journal of Recent Trends in Engg., Vol. 1, No. 2, May 2009.

[4] Cao G. and Singhal M., "*On Coordinated Checkpointing in Distributed Systems*", IEEE Trans. on Parallel and Distributed Systems, Vol. 9, No.12, pp. 1213-1225, Dec.1998.

[5] Cao G. and Singhal M., "*Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems*", IEEE Trans. on Parallel and Distributed Systems, Vol. 12, No.2, pp. 157-172, Feb. 2001.

[6] Prakash R. and Singhal M., "*Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems*", IEEE Trans. on Parallel and Distributed Systems, Vol. 7, No.10, pp1035-1048, Oct. 1996.

[7] Pradhan, D.K., Krishna, P., and Vaidya, N.H. *" Recovery in mobile environments: Design and trade-off analysis*". In Proceedings of the 26th International Symposium on Fault-Tolerant Computing, (Sendai, Japan, June 1996), IEEE, pp. 16–25.

[8] Kumar P., Kumar L., Chauhan R.K. and Gupta V.K., "*A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for mobile Distributed Systems*", in the Proc. of the IEEE ICPWC-2005, Jan. 2005.

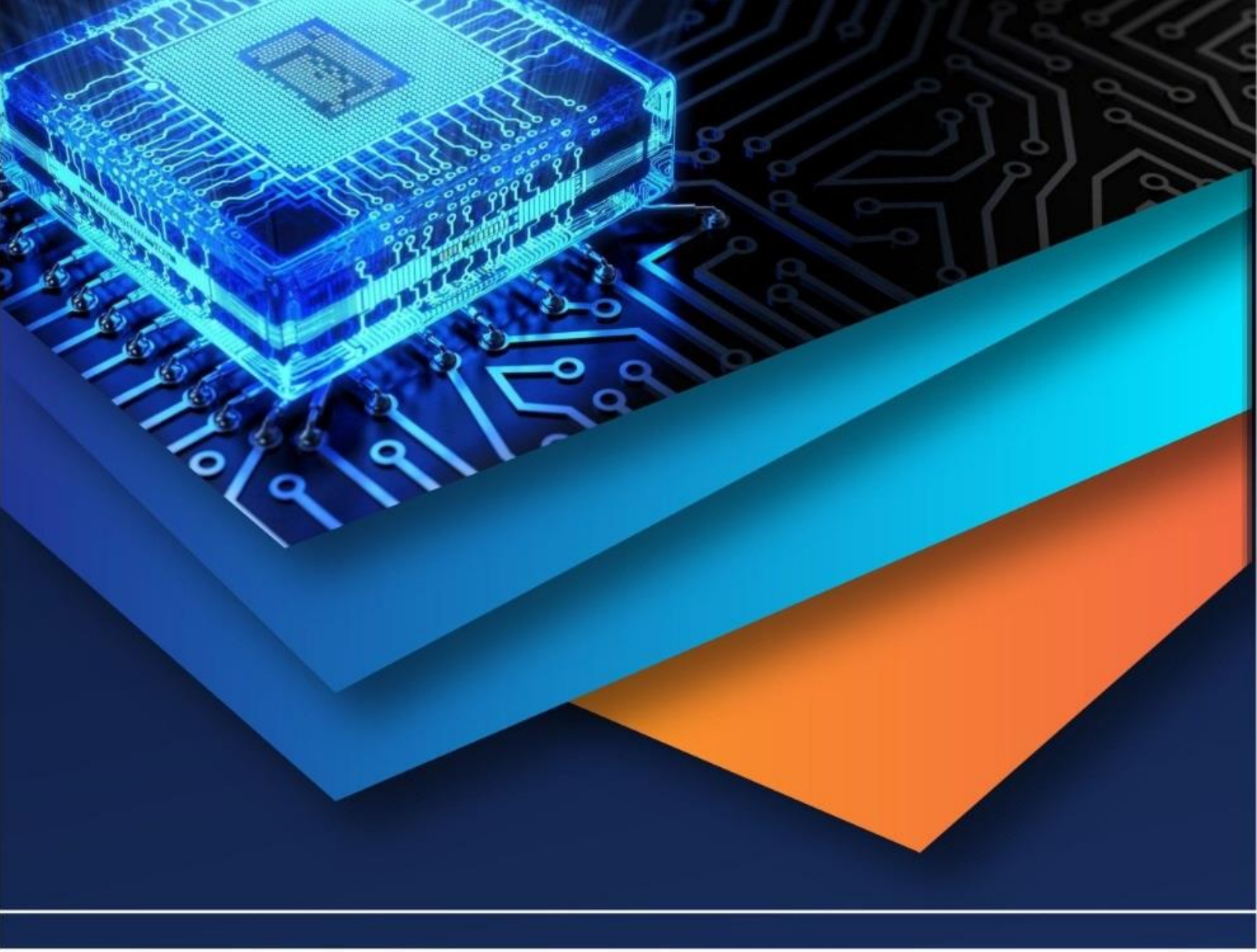
[9] Randell, B. System "*structure for software fault tolerance*". IEEE Trans. Softw. Eng. SE-1, 2 (June 1975), 220–232.

[10] N. Neves, "*Time-based coordinated checkpointing*," Ph.D. dissertation, UIUCDCSR- 98-2054, University of Illinois at Urbana- Champaign, 1998.

[11] P. Kumar and R. Garg, "*Soft checkpointing based coordinated checkpointing protocol for Mobile Distributed Systems*", International Journal of Computer Science Issues, Vol. 7, Issue 3, No. 5, May, 2010.

[12] S. Kumar, R.K. Chauhan and P. Kumar,"*Reliable Soft-Checkpoint Based Fault Tolerance Approach for Mobile Distributed Systems*", International Journal of Computer and Network Security", Vol. 2, No., June, 2010.

[13] Parveen Kumar, R K Chauhan, "*A Coordinated Checkpointing Protocol for Mobile Computing Systems*", International Journal of Information and Computing Science, Vol. 9, No. 1, pp. 18-27, 2006.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY