Real Time Snapshot Collection Algorithm for Mobile Distributed Systems with Minimum Number of Checkpoints

Surender Kumar
Deptt. of Information Technology, HCTM, Kaithal

R.K. Chauhan
Deptt. of Computer Sc. & Application, Kurukshetra University, Kurekshetra

Parveen Kumar
Deptt. of Computer Sc. & Engg. MIET, Meerut

ABSTRACT
Checkpointing is an efficient way of implementing fault tolerance in distributed systems. Mobile computing raises many new issues, such as high mobility, lack of stable storage on mobile hosts (MHs), low bandwidth of wireless channels, limited battery life and disconnections that make the traditional checkpointing protocols unsuitable for such systems. Minimum process non-blocking coordinated checkpointing may be useful for mobile distributed system as this approach is domino-free, requires at most two checkpoints of each process on stable storage, forces only interacting processes to checkpoint and does not suspend their underlying computation during checkpointing. Sometimes, it also requires piggybacking of information onto normal messages, blocking of the underlying computation or taking some useless checkpoints. In this paper, we propose a non-blocking minimum process coordinated checkpointing algorithm that requires minimum bandwidth over wireless channels and does not requires any induced/forced or mutable checkpoints and reduce the height of checkpointing tree without taking any extra overhead in real time.

Keywords
Fault tolerance, checkpointing, consistent global state, domino free, orphan message, coordinated checkpointing and mobile distributed systems.

1. INTRODUCTION
Checkpointing is a well-established technique used for fault-tolerance in distributed systems. To recover from a failure, the system restarts its execution from a previous error-free, consistent global state [3]. A global state is said "consistent" if it contains no orphan message (whose receive event is recorded but its sent event is lost).

Coordinated checkpointing is a commonly used technique for fault tolerant [1], [4], [6], [8]-[9], [13]-[15] in mobile distributed system, as it is domino free. In coordinated checkpointing, processes must coordinate their checkpointing activities and take checkpoints in such a manner that the resulting global state is consistent. The Chandy-Lamport [6] algorithm is the earliest non-blocking all-process coordinated checkpointing algorithm.In this algorithm a marker are sent along all channels in the network and requires FIFO channels.

Koo-Tong [4] have proposed a minimum process algorithm which use sequential coordinated scheme and block their relevant processes during checkpointing. In algorithm [8] and [15] authors proposed a non sequential minimum process algorithm but it also blocks their underlying computation as [4]. Further to remove blocking overhead in [6] authors proposed all process non- blocking centralized checkpointing algorithms with minimum synchronization message overhead.

Recently, non-blocking distributed checkpointing algorithms [14] have received consideration attention. However, the algorithm [14] also forces all processes as [4] and [13], even though many of them may not be necessary. The Parkash-Singhal [9] proposed the first minimum process non blocking checkpointing algorithm. This algorithm only forces the minimum number of processes to take checkpoints without blocking of the underlying computation. However author found paper [8], that algorithm [9] may leads inconsistency in some situation and proved that there does not exists a non-blocking algorithm which forces only a minimum number of processes to take their checkpoints. Cao and Singhal [1] achieved non-intrusiveness in the minimum-process algorithm by introducing the concept of mutable checkpoints. The number of useless checkpoints in [1] may be exceedingly high in some situations [11].

The rest of the paper is organized as follows. Section II presents the system model, section III formulate study and inconsistency in existing checkpointing algorithms, section IV presents basic idea, feature, data structure etc, section V presents main algorithm, section VI show example, section VII presents correctness proof, VIII presents performance evaluation and section VIII presents comparative study and conclusions.

2. SYSTEM MODEL
A mobile system is a distributed system where some of processes are running on mobile hosts (MHs) [5].The term "mobile" means able to move while retaining its network connection. A host that can move while retaining its network connection is an MH (see Figure 1). 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 through the base station(BS) .A cell is a geographical area around an base station 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 .An MSS has both wired and wireless links and acts as an interface between static network and a part of mobile network . Static network connects all MSSs.

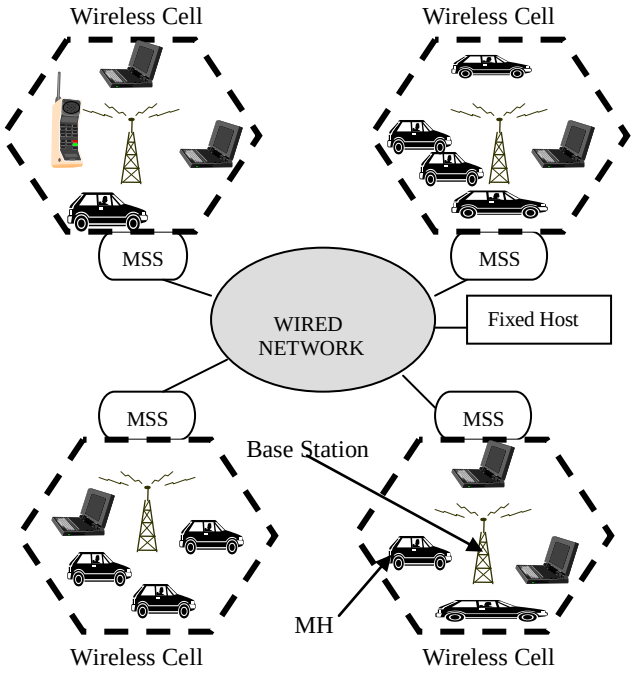

Figure 1. Mobile Distributed System

Our system model is similar to [1] and 11]. There are n spatially separated sequential processes denoted by P_0, P_1,.., P_{n-1}, running on MHs or MSSs, constituting a mobile distributed computing system. Each MH/MSS has one process running on it. The processes do not share memory or clock. Message passing is the only way for processes to communicate with each other. Each process progresses at its own speed and messages are exchanged through reliable channels, whose transmission delays are finite but arbitrary. A process is in the cell of MSS means the process is either running on the MSS or on an MH supported by it. It also includes the processes of MHs, which have been disconnected from the MSS but their checkpoint related information is still with this MSS. We also assume that the processes are non-deterministic and FIFO channels. In our algorithm mobility and disconnections are handled as in algorithm [1]

3. EXISTING COORDINATED CHECKPOINTING ALGORITHM

Many checkpointing algorithms have been proposed for the distributed as well as mobile distributed systems. Some checkpointing algorithm forces minimum process to take their checkpoint but must block their underlying computation during checkpointing as in [4]. Blocking algorithms may dramatically degrade the system performance [6]. To increase the system performance non-blocking checkpointing algorithms are proposed. This non-blocking checkpointing algorithm uses checkpoint sequence number (csn) to identify the orphan message. However, these algorithms requires useless checkpoint during checkpointing, even though many of them may not be necessary.

The algorithm in [9] was the first coordinated non-blocking algorithm that tries to combine min-process with non-blocking two approaches. It only forces minimum number of processes on which initiator depends directly or transitively to take checkpoints and does not block their underlying computation during checkpointing.

However in [8], the author point out that the algorithm in[9] can cause inconsistencies in some situations and proposed new checkpointing algorithm to correct the inconsistency.

Problem with Cao and Singhal's Algorithm [1]

In [1] Cao and Singhal proposed mutable checkpoints based checkpointing algorithm to improve in[9] and implement a non blocking checkpointing algorithm. In Cao and Singhal's algorithm [1], when P_i receives a computation message M from P_j , P_i take mutable checkpoint if the following three condition have been satisfied.

i. P_j is in checkpointing process before sending M.
ii. P_i has sent a message since last checkpoint.
iii. P_i has not taken a checkpoint associate with current initiation.

Figure 2, shows the inconsistency exists in Cao and Singhal's algorithm [1] by the example in 3.4. in Figure 2,

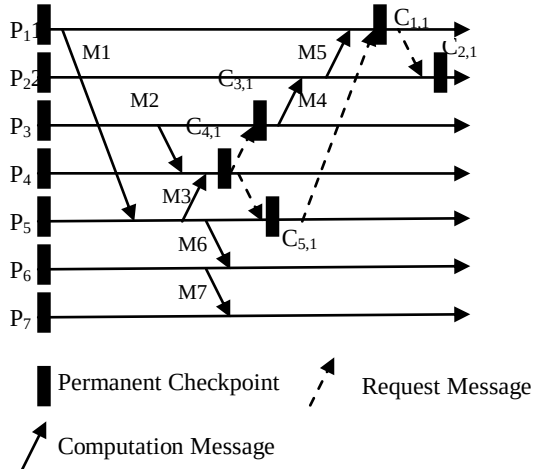

Figure 2. An example showing inconsistency in algorithm [1]

P_4 initiates the checkpointing algorithms, takes the tentative checkpoints $C_{4,1}$ and sends the checkpoint request message to processes P_3 and P_5, as it depends on these. So, P_3 and P_5 take tentative checkpoints after receiving the request.

After taking checkpoint $C_{3,1}$, P_3 sends M4 to P_2. P_2 doesn't

take mutable checkpoint before delivering M4 because it hasn't send a message since last checkpoint (condition(2) false). After receiving M5 from P_2, P_1 receives checkpoint request from initiator and request P_2 to take checkpoint further. So M4 become orphan message.

4. PROPOSED CHECKPOINTING ALGORITHM

4.1 Basic Idea

We propose a two phase checkpointing algorithm to remove the inconsistency. In the first phase checkpoint initiator compute the minset, take tentative checkpoint and sends the checkpointing request to all the process in minset with weight. Upon receiving checkpoint request process Pi takes tentative checkpoint (if not taken any forced or tentative checkpoint related to current initiation), convert forced checkpoint into tentative one (if taken forced checkpoint related to the current initiation) or ignore the request (if already taken the tentative checkpoint) and increases csni[i]. After taking the tentative checkpoint, P_i propagates the checkpoint request to all processes P_k further, which are dependent processes and not belongs to minset by appending initiator's trigger and a portion of received weight. At last, P_i sends a reply to the initiator with the remaining weight and continues computation.

In second phase initiator broadcast COMMIT or ABORT message. When the initiator P_{jni} receives a reply message from the processes in minset, P_{ini} adds the weight which is in the reply message to its own weight. When the weight becomes equal to 1, it conclude that all processes involved in checkpointing have taken their tentative checkpoints successfully. Then, it broadcasts COMMIT message. On receiving the COMMIT message process P_i , if a process has taken a tentative checkpoint, it convert tentative checkpoint to permanent; if a process has taken a forced checkpoints, it discard forced checkpoint and decreases the $csn_i[i]$. Each process updates its csn and other data structure according to the piggybacked on committing message. On the other hand if weight is not equal to 1 and time out, then P_{ini} broadcast ABORT message. After receiving the ABORT message process P_i, rollback to its previous consistent state.

Sending and Receiving Computation Message during Checkpoinitng:

When a process P_i in checkpointing session sends a computation message to process P_j , it piggybacks his csn, trigger, and minset with the message.

On the receiving end following actions are taken:

a) if $(old\_csn_j[i] >= m.csn_i[i])$*:* it means both the processes takes latest checkpoint related to the current initiation. So in such case process only receive the message and updates the data structure.

b) if $(old\_csn_j[i] < m.csn_i[i])$*:* in such case the following actions are taken

(i) if $(P_i \in minset[])$*:*takes tentative checkpoint.(as process is a part of minset and definitely get the checkpointing request from the initiator).

(ii) if $((P_i \notin minset[]) \wedge$ *(Bitwise logical AND of sendvi[]* $\wedge$ *minset[] is not all zero)):* process does not belongs to minset and send any computation message to the processes which belongs to minset, since its last checkpoint, it takes tentative checkpoint (as there is a good probability that process will get the checkpoint request).

(iii)) $if\ ((Pi \notin minset[]) \wedge$ *(Bitwise logical AND of sendvi[]* $\wedge$ *minset[] is all zero)) :* in such case there is a probability that a process do not get any checkpoint request process takes the forced checkpoint

4.2 Data Structure

Each process Pi maintains the following data structures:

P_{int}	Initiator process identification
m.csn	senders csn received with message
m.g_set	sender global set received with message
$weight_i$	A non negative real variable with a maximum value of 1and used to value 1 used to detect the termination of checkpointing algorithm as in [7].
mr_i	A flag set to "1" on taking the tentative checkpoint successfully.
$csn_i[]$	An array of length n for n processes at each process P_i, where $csn_i[j]$ indicates the checkpoint sequence numbers (csn) of P_j currently known to P_i.
old_csn_i	The csn of P_i's last checkpoint.
$ddv_i[]$	A bit vector of size n; $ddv_i[j]$ =1 implies P_i is directly dependent upon P_j for the current CI; initially, k, $ddv_i[k]$=0 and $ddv_i[i]$=1;
$Sendv_i[]$	A bit vector size n; $sendv_i[j]$=1 implies P_i has sent at least one message to P_j in the current CI.
minset[]	A bit vector of size n which is compute on the MSS_{ini}; if P_i initiate its(x+1)th checkpoint then the set of processes on which P_i depends (directly or transitively) in its xth checkpoint interval is minimum set. In order to compute the initial minimum set we use the similar approach as [9].
new_ddv_i	It holds the new dependency at node P_i during the execution of checkpoint request.
Uminset[]	It holds exact minimum set at last.
c_req	Checkpoint request send by a process.
c_rply	After receiving the checkpoint request, processes sends reply (acknowledge) negatively or positively to initiator process.
g_set	A set of 2-tuples(Pid,csn) maintained by each process P_i, where Pid indicates the identifier of a checkpointing initiator and csn indicates the checkpoint sequence number at process Pid corresponding to the checkpoint event as [12].

4.3 Maintains of Dependency Vector

In order to maintain the dependency vector $ddv_i[]$, we use the similar approach as the [9], where each process Pi maintains a Boolean vector $ddv_i[]$, which has n bits. Initially at Pi, the vector $ddv_i[]$set to 0 except Pi[i] and set

$ddv_i[j]$ to '1' only if Pi receive computation message(m) from Pj. So, $ddv_i[j]$ =1 represents that Pi is directly dependent upon Pj for the current CI.

When process Pi sends a computation message m to Pj, it appends $ddv_i[]$ to m (see Figure 2). After receiving m, Pj includes the dependences indicated in $ddv_i[]$ into its own ddvj[] as follows: $ddv_j[k] = ddv_j[k]$ v m. ddv[k] , where 1<=k<=n, and v is the bitwise inclusive OR operator. Thus, if a sender Pi of a message depends on a process Pk before sending the computation message, the receiver Pj also depends on Pk through transitivity. So in this way ddv[] contain all the processes which are directly or transitively dependent on the process. The dependency information is used to minimize the effort required to collect global checkpoint.

Minimum set is a bit vector of size n which is compute by the MSSini by taking transitive closure of dependency of dependency bit vector with its own dependency bit vector. So at the time of initiation ddv [] of the MSSini treated as a minimum set. (minset[]= $ddv_{ini}[]$). minset[k]=1 implies Pk belongs to the minimum set and it is directly or transitively dependent on initiator process Pini.

4.4 Mobility and Disconnections

Due to mobility a MH may disconnect from the old MSS and connected to a new MSS. Due to this message transmission becomes complicated. In paper [10] routing protocol has been proposed to handle the MH mobility. Disconnection may me voluntary or non-voluntary. We can handle the voluntary disconnection and non-voluntary disconnection are treated as faults [1],[3].

Suppose, an MH, say MHi, disconnects from the MSS, say MSSk. MHi takes its checkpoint, say disconnect_ckpti, and transfers it to MSSk. MSSk stores all the relevant data structures and disconnect_ckpti of MHi on stable storage. If MHi is in the minset[], disconnect_ckpti is considered as MHi's checkpoint for the current initiation. On commit, MSSk also updates MHi's data structures, e.g., ddv[], send, etc. On the receipt of messages for MHi, MSSk does not update MHi's ddv[], but maintains a message queue to store the messages.

When MHi enters in the cell of MSSj, it is connected to the MSSj if no checkpointing process is going on. Before connection, MSSj collects its ddv[], buffered messages, etc. from MSSk; and MSSk discards MHi's support information and disconnect_ckpti. The stored messages are processed by MHi, in the order of their receipt at the MSS. MHi's ddv[] is updated on the processing of buffered messages. If a node does not reconnect in a stipulated time, then its computation can be restarted from its disconnect_ckpt.

5. The Algorithm

Actions for the Initiator/Coordinator Pg

a) *on checkpoint initiation:*

{set new_set[]= = 0; Uminset[]= = minset[];
c_state_i=1; csn_i= old_csn_i +1; $weight_i$=1.0;
set g_set(P_g.pid, P_g.csn);
check ddv_i ;
when $ddv_{ini}[k]$ = =1 for 1<=k<=n;
set minset[k]=1;}

b) *sends c_req() to all node P_j such that and wait for response:*

for(j=0;j<=n; j++)
if (minset[j]= = 1)
$weight_i$ = $weight_i$/2; ws=$weight_i$;
sends c_req(g_set, minset, ws, request);

c) *on receiving response from process Pj:*

receive message c_rply (new_ddv_j[],w_j, mr)
$weight_i$ = $weight_i$ + w_j ;//update weight
if(new_ddv_j[] ≠∅)
{new_set = new_set[] U new_ddv_j[]; //update new_set
Uminset = minset[] U new_set[];}//update Uminset

d) *send ABORT or COMMIT message:*

if((weight<1) AND(maxtimeout)) ∨ (mr==0)
{sends message ABORT() to all processes belongs to Uminset[]}
else if(weight= =1)
{send message COMMIT() to all process belongs to Uminset[];}

Actions taken when Pj sends Computation Message to Pi

Set $sendv_j$[i] =1;
If tentative // message sends after taking tentative checkpoint
Send(P_i, msg, ddv_j[], csn_j[j], minset[], g_set)
else
Send(P_i, msg, ddv_j[],csn_j[j], ∅ , ∅);

Actions taken when Pi receive Computation Message from Pj

Receive msg(P_i, msg, ddv_j[], csn_j[j], minset[], g_set);

a) if($old\_csn_j > m.csn_i[j]$)
{Receive(msg) and update csn_i[j] and ddv_i[j];}

b) if($old\_csn_j < m.csn_i[j]$) ∧ ($m.g\_set=\emptyset$)
{Receive(msg) and update csn_i[j] and ddv_i[j];}

c) if($old\_csn_j < m.csn_i[j]$) ∧ ($m.g\_set \neq \emptyset$)

i) *if($P_i \in$ minset[]) ∧ (own.g_set ≠ m.g_set)*
{Take tentative checkpoint; receiving message; increment old_csn_i; set Master; update csn_i[j];update ddv_i[j]; }

ii) if($P_i \in$ minset[]) ∧ (own.g_set == m.g_set)
{Receive(msg) and update csn_i[j] and ddv_i[j];}

iii) if (($P_i \notin$ minset[]) ∧ (Bitwise logical AND of $sendv_i[] \wedge$ minset[] is all zero))
{Take forced checkpoint}

iv) *if (($P_i \notin$ minset[]) ∧ (Bitwise logical AND of $sendv_i[] \wedge$ minset[] is not all zero))*
{Take tentative checkpoint; receiving message; increment old_csn_i; set Master; update csn_i[j];update ddv_i[j]; }

v) *if* ($P_i \notin minset[]) \wedge$ ($sendv_i[] = = \varnothing$)
{receive message; }

Actions taken when Pj receives checkpoint request from Pi and forward to Pk

receive c_req(g_set, minset, ws, request) ;
if (req.g_set=own.g_set)
{ P_j ignore the request and sends reply message to P_i with received weight ; }
else
{ Take tentative checkpoint ; increment csn_j ; check ddv_j[];
For(k=0; k<n; k++)
if($\forall$ *k s.t.* $ddv_j[k] = =0) \vee (\forall$ *k s.t.* $ddv_j[k] = =1 \wedge$ *minset[k] = =1)*
{wr = ws;
Sends message c_rply($\varnothing$, wr, mr,P_j) to initiator,
Continue computation ;
}
else if($\exists$ *k s.t.* $ddv_j[k] = =1 \wedge$ *minset[k] = =0)*
if($sendv_k[] = = \varnothing) \vee (\exists$ *p s.t.* $sendv_k[p] = =1 \wedge$ *minset[p] = =0)*
{Set new_ddv_j ==1;
wr = wr/2; ws=wr;
sends c_req(minset,csn_j, req.g_set,ws, req) to P_k ;
sends c_rply(new_ddv_j[k],wr, mr) to P_i;
continue computation;
}
}

When Process Pj receives COMMIT/ABORT message:

On receiving COMMIT ( )

{Discard old permanent checkpoint, if any;
convert tentative checkpoint in to permanent,
Reset the related data structure}

On receiving ABORT ( )

{Discard the tentative checkpoint; and reset data structure}

6. AN EXAMPLE

We explain our checkpointing algorithm with the help of an example. Consider the distributed system as shown in Figure 3. Note that when a computation message is sent after taking the checkpoint it piggybacked with minset[]. Assuming that process P_4 initiate checkpointing process. First process P_4 takes its tentative checkpoint and increment its csn number from $C_{4,0}$ to $C_{4,1}$, compute minset[](which in case of Figure 3. is {P_1, P_3, P_5}). This means is that the initiator process is directly or transitively dependent on these processes. Hence, when P_2 initiate a checkpoint all of these processes should take their checkpoints in order to maintain global consistent state. Therefore P_2 sends the checkpoint request along with minset[] to process P_1, P_3 and P_5. When P_3 receives the checkpoint request it takes the tentative checkpoint and sends message M4 by atteching minset [1011100], trigger set(P_4, $C_{4,1}$), and csn_3=1. After receiving message M4, P_2 first compare $m.csn_3$(which is 1) with its old_csn_2[3](which is 0). As P_2 does not belongs to minset, not sent any message to the processes which are in minimum set and $m.csn_3 > csn_2[3]$. Hence, P_2 takes forced checkpoint, update its trigger set to (P_4, $C_{4,1}$), increment its csn_2 from 1 to 2, and updates the csn_2[3] from $C_{4,0}$ to $C_{4,1}$.

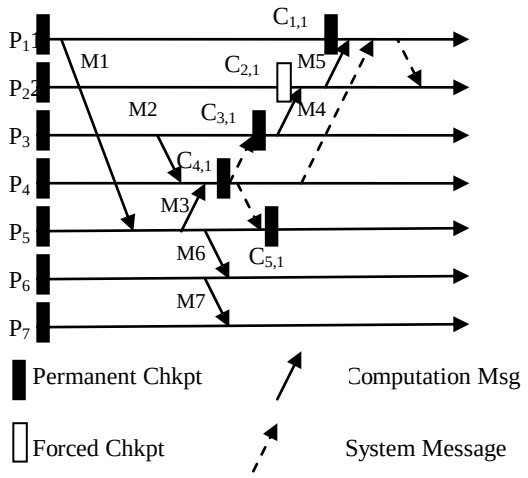

Figure 3. An example showing inconsistency in algorithm [1]

After taking forced checkpoint it sends message M5 to P_1. P_1 takes tentative checkpoint directly due to minset[P_1]= =1 and set c_state ==1(as P_1 knows that it is the part of minset and get the checkpoint request from the initiator in future and when it get the checkpoint request it ignore the request).P_2 check its dependency and find out that it receives computation message from P_2 since its last checkpoints. So, it sends checkpoint request to the process P1 with weight and reply with remaining weight and new_ddv_2 [P1] ==1 to the initiator. After receiving the checkpointing request from P_1, P_2 converts its forced checkpoints in to tentative one and reply to the initiator. Initiator compute the Uminset[P_1, P_2, P_3, P_4, P_5] by taking the union of minset{ P1, P3, P4, P5} and new_ddv_1{P_2}.
At last, when P_2 receives positive responses from all relevant processes(weight = =1) it issues commit request along with the exact minimum set [P_0, P_1, P_2, P_3, P_4] to all processes. On receiving commit following actions are taken. A process, in the minimum set, converts its tentative checkpoint into permanent one and discards its earlier permanent checkpoint, if any. On the other hand if it receive the negative response from any one of the processes which belongs to the minset, it sends the abort message to all processes which belongs to Uminset[]. On receiving abort, processes discard the tentative checkpoint, if any; reset c_state, tentative, g_chkpt etc and update ddv[] and minset[].The system is consistent.

7. CORRECTNESS PROOF

Let GCi ={C1,x, C2,y,.....,Cn,z} be some consistent global state created by our algorithm, where Ci,x is the xth checkpoint of Pi.

Theorem I: Algorithm is non-blocking and produces a consistent global state.

Proof: Processes which are part of global state can receives

and handle the computation messages by the following ways

a) Message received from the processes before sending any ddv[] with message to the initiator or any other processes which are directly or transitively depends upon initiator: These types of processes become the part of the minset and receives the checkpoint request directly from the initiator so that these messages not become orphan.
b) Message received from the processes after sending the ddv[] with message to initiator the initiator or any other processes which are directly or transitively depends upon initiator but before receiving the checkpoint request and taking tentative checkpoint. Message received after taking tentative checkpoint and before receiving the commit request: These types of message are buffered and execute after the checkpoint interval.

In such way there are not any orphan message and handle all messages efficiently without blocking. So it shows that our algorithm is non-blocking and produces the global consistent state.

8. PERFORMANCE EVALUATION OF PROPOSED CHECKPOINTING ALGORITHM

To evaluate we compare the performance of our proposed minimum process checkpointing algorithm with [1], [2], [4], [6], [14] in different perspective. We assume an n+1 process distributed system and use the following notations for performance analysis of the algorithms:

N : Total number of processes.
N_{min}: Minimum number of processes that required to take checkpoint.
N_{mut}: Number of redundant Mutable checkpoint during a checkpointing process.
N_{indu} Number of redundant Induced checkpoint during a checkpointing process
C_{broad}: Cost of broadcasting a message to all (N) processes in the system.
C_{air} : Cost of sending a message from one process to another process.
T_{ch}: Total checkpointing time. This time includes the time to save the checkpoint on MSS, transferring time from MH to its MSS and times taken by a system message during a checkpointing process.

8.1 Performance of our Checkpointing Algorithm

The Blocking Time: Similar to algorithms [1], [2], [6] and [14], our algorithm does not block their underlying computation during checkpointing.

The Number of Checkpoints: Similar to algorithms [1], [2], [4], our algorithm also forces only a minimum number of processes to take their checkpoints.

The Average Message Overhead: our algorithm includes the following message overhead in best case is given as $3*N_{min} * C_{air}$ in table 1. In our algorithm, first the initiator sends control messages to minimum number of processes that need to take a checkpoint each and reply (acknowledge) back. At last when initiator receives acknowledge from all the processes, it sends COMMIT message to these minimum processes to convert their respective tentative checkpoint in to permanent one. Hence total cost of these are $3*N_{min}*C_{air}$.

Useless Checkpoints: Our algorithm does not have any useless checkpoints as [1] and [2]. Instead of above, our algorithm is coordinated, nondeterministic, distributed and require piggybacking of integer csn(checkpoint sequence number) on normal messages .

8.2 Comparison with Existing Algorithms

In [1], Cao-Singhal proposed a mutable checkpoint based non-blocking minimum-process coordinated checkpointing algorithm. This algorithm completes its processing in the following three steps. First initiator MSS sends tentative checkpoint request to minimum number of processes that need to take checkpoint. The synchronization message overhead for this is $N_{min} * C_{air}$. Secondly MSS_{ini} gets the acknowledgement from all processes to whom it sent checkpoint request. Hence message overhead $2* N_{min} * C_{air}$ is needed in first two phases. At last MSS_{ini} sends the COMMIT request to convert its tentative checkpoint into permanent. In this case it takes min ($N_{min}* C_{st}$, C_{broad}). Hence algorithm [1] generate consistent global state with the message overhead cost $2* N_{min} * C_{air} + min (N_{min}* C_{air}, C_{broad})$ and average number of checkpoints $N_{min}+ N_{mut}$ [Refer Table 1]. Thus algorithm is non-blocking and minimum process but suffer from useless checkpoints. Our proposed algorithm generates the consistent global state with approximately same message overhead as [1], without using any useless checkpoint. In [2], P.Kumar et al. also proposed minimum process coordinated checkpoint algorithm for mobile system. The synchronization message overhead to complete the checkpointing process using algorithm [2] is given as $3*C_{broad} + 2*N_{min} * C_{air.}$ Here $3C_{broad}$ is the total cost of broadcasting sends ddv[](C_{broad}), take tentative checkpoint the request(C_{broad}) and COMMIT(C_{broad}) messages to all MSSs by the initiator MSS. $2*N_{min}*C_{air}$ is the total cost of sending checkpoint request message to the minimum number of processes that need to take checkpoints($N_{min}*C_{air}$) and reply to the initiator after taking the tentative checkpoint($N_{min}*C_{air}$). Hence algorithm [2] generates the global consistent state by using $N_{min}+ N_{indu}$ average number of checkpoints and $3*C_{broad} + 2*N_{min} * C_{air}$ message overhead cost but our proposed algorithm by using N_{min} and $3* N_{min} * C_{air}$ respectively [Refer Table 1].Thus algorithm [2] takes less useless checkpoint in the comparison of [1] but have high message overhead cost. The algorithm suffers from useless checkpoint and has higher message overhead as compared to the proposed algorithm. The koo-Toueg[4] proposed a minimum process coordinated checkpointing algorithm for distributed systems with the cost of blocking of processes during checkpointing.This algorithm

Table 1.										
Our Algo.	$3*N_{min} * C_{air}$	N_{min}	0	No	Yes	Integer	No	Yes	Yes

	Cao-Singhal [8]	Parveen Kumar [15]	S.K.Gupta [16]	Cao-Singhal [1]	P.Kumar et al. [2]	Koo-Tong [4]	Elnozahly [6]	S.Neogy [14]
Average Message Overheads	AOM_{minp}	AOM_{minp}	$2*N_{min}*Cpp+C_{broad}$	$2*N_{min}*C_{air}+min(N_{min}*C_{air},C_{broad})$	$3*C_{broad}+2*N_{min}*C_{air}$	$3*N_{min}*N_{dep}*C_{air}$	$2*C_{broad}+N*C_{air}$	$2*C_{broad}+N*C_{air}$
Avg. no. of Checkpoint	N_{min}	N_{min}	N_{min}	$N_{min}+N_{mut}$	$N_{min}+N_{indu}$	N_{min}	N	N_{mss}
Avg. Blocking Time	$2T_{st}$	$2T_{st}$	0	0	0	$N_{min}*T_{ch}$	0	0
Useless Checkpoint	No	No	No	Present	Present	No	No	No
Minimum Processes	yes	yes	Yes	Yes	Yes	Yes	No	No
Piggyback Info.	Nil	Integer	Integer	Integer	Integer	Integer	Integer	Integer
Concurrent Execution	No	No	No	Yes	No	No	No	No
Distributed	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Non-deterministic	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes

requires minimum number of synchronization message and number of checkpoint .In Toueg algorithm requires only minimum number of process to take checkpoints (ii) message overhead is $3*N_{min}*N_{dep} * C_{air}$ (iii) Blocking time is $N_{min}*T_{ch}$. Our proposed algorithm reduces the message overhead $3*N_{min}*N_{dep} * C_{air}$ *to* $3*N_{min}* C_{air}$ [Refer Table 1]. Thus algorithm [2] takes less useless
checkpoint in the comparison of [1] but have high message overhead cost. The algorithm suffers from useless checkpoint
and has higher message overhead as compared to the proposed algorithm. In [6] and [14] authors designs an all process non blocking checkpointing algorithm. In these algorithms we get consistent global state with the total cost of $(2*C_{broad} + N*C_{air})$ [Refer Table 1].
However algorithm [6] and [14] had fewer messages overhead in the comparisons of our proposed algorithm but these algorithms forces to all processes in the system to take their checkpoints for each checkpoint initiation. This may waste the energy and processor power of the processes which are in doze mode. Compared to [6], our algorithm forces only a minimum number of processes to take checkpoint on stable storage.
In Elnozhay et al.[6] and S.Neogy et al.[14] algorithm proposed non blocking checkpointing algorithms but requires all-processes to take checkpoints during checkpointing, even though many of them may not be necessary. In mobile environment, since checkpoints need to be transferred to the stable storage at the MSSs over the wireless network. So in this way taking unnecessary checkpoints may waste a large amount of wireless bandwidth. In the algorithms [4] and [8] authors proposed minimum process checkpointing algorithm but it block its underlying computation during checkpointing. The blocking time of the Koo-Toueg[4] (N_{min}* T_{ch})algorithm is highest, followed by Cao-Singhal[8] which is $2T_{st}$ (not shown in the
table1). Therefore, blocking algorithm may degrade the performance to mobile computing systems [6].
The message overhead in proposed algorithm is greater than [6] but less than [1]. However, the algorithm in [6] is a centralized algorithm and there is no easy way to make it distributed without increasing message overhead. The Parkash-Singhal[9] proposed the first minimum process non blocking checkpointing algorithm. However author found that this algorithm may result in an inconsistency [3]and [8] in some situation and proved that there does not exist a non-blocking algorithm which forces only a minimum number of processes to take their checkpoints. Cao and Singhal [1] achieved non-intrusiveness in the minimum-process algorithm by introducing the concept of mutable checkpoints but number of useless checkpoints in [1] may be exceedingly high in some situations [11]. Also a concurrent execution is allowed in [1], but in algorithm [12], author proves that algorithm [1] may lead to inconsistency during concurrent execution. Kumar et. al [2] proposed a five phase checkpointing algorithm to reduced the height of the checkpointing tree and the number of useless checkpoints by keeping non-intrusiveness intact. However, algorithm [2] reduces the useless checkpoint in the comparison of algorithm [1] but has extra message overhead cost.

9. COMPARATIVE STUDY AND CONCLUSION

In Elnozhay et al.[6] and S.Neogy et al.[14] algorithm proposed non blocking checkpointing algorithms but requires all-processes to take checkpoints during checkpointing, even though many of them may not be necessary. In mobile environment, since checkpoints need to be transferred to the stable storage at the MSSs over the wireless network. So in this way taking

unnecessary checkpoints may waste a large amount of wireless bandwidth. In the algorithms [4] and [8] authors proposed minimum process checkpointing algorithm but it block its underlying computation during checkpointing. The blocking time of the Koo-Toueg[4] (N_{min}* T_{ch})algorithm is highest, followed by Cao-Singhal[8] which is $2T_{st}$ (not shown in the table1). Therefore, blocking algorithm may degrade the performance to mobile computing systems [6]. The message overhead in proposed algorithm is greater than [6] but less than [1]. However, the algorithm in [6] is a centralized algorithm and there is no easy way to make it distributed without increasing message overhead. The Parkash-Singhal[9] proposed the first minimum process non blocking checkpointing algorithm. However author found that this algorithm may result in an inconsistency [3]and [8] in some situation and proved that there does not exist a non-blocking algorithm which forces only a minimum number of processes to take their checkpoints. Cao and Singhal [1] achieved non-intrusiveness in the minimum-process algorithm by introducing the concept of mutable checkpoints but number of useless checkpoints in [1] may be exceedingly high in some situations [11]. Also a concurrent execution is allowed in [1], but in algorithm [12], author proves that algorithm [1] may lead to inconsistency during concurrent execution. Kumar et. al [2] proposed a five phase checkpointing algorithm to reduced the height of the checkpointing tree and the number of useless checkpoints by keeping non-intrusiveness intact. However, algorithm [2] reduces the useless checkpoint in the comparison of algorithm [1] but has extra message overhead cost.

Since a forced checkpoints are save in main memory, the delay incurred in saving a forced checkpoints, is very little in the comparison of save a tentative checkpoints on stable storage. Our proposed algorithms also try to minimize the useless checkpoints. Hence, proposed coordinated checkpointing algorithms obtain a consistent global checkpoint state by minimizing the number of additional checkpoints, forcing only minimum processes and without blocking. It has also the low overhead in the comparison of other algorithms. [Table 1]

10. REFERENCES

1. Cao G. and Singhal M., Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems. IEEE Transaction On Parallel and 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, January 2005.
3. Cao-Singhal : On coordinated checkpointing in Distributed Systems. IEEE Trns. on Parallel and Distributed Systems, vol. 9, no.12, pp. 1213-1225, Dec 1998.
4. Koo-Toueg.: Checkpointing and Roll-back Recovery for Distributed systems. IEEE Transactions on Software Engineering, pages 23-31, January 1987.
5. Acharya A. and Badrinath B. R.,: Checkpointing Distributed Applications on Mobile Computers. Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, pp. 73-80, September 1994.
6. Elnozahy E.N., Johnson D.B. and Zwaenepoel W.: The Performance of Consistent Checkpointing, Proceedings of the 11th Symposium on Reliable Distributed Systems, pp. 39-47, October 1992.
7. Hung, S. T.: Detecting Termination of Distributed Computations by External Agents, Proceeding 9th Int' Conf. Dist, Computing System, pp.79-84, 1989.
8. Cao-Singhal. : On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems, Proc. of Int'Conf. on Parallel Processing, pp. 37-44, August 1998.
9. Prakash R. and Singhal M.: Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems, IEEE Transaction On Parallel and Distributed Systems, vol. 7, no. 10, pp. 1035-1048, October1996.
10. C. Perkins : Mobile IP, IEEE Comm. Magazine, vol. 35, pp. 84-99, May 1997.
11. L. Kumar, M. Misra, R.C. Joshi :Low overhead optimal checkpointing for MDS, Proc.19th IEEE Int'Conf on Data Engineering, pp 686 – 88, 2003.
12. Ni, W., S. Vrbsky and S. Ray: Pitfalls in Distributed Nonblocking Chkpointing, J' of Interconnection N/Ws, Vol. 1 No. 5, pp. 47-78, March 2004.
13. L.Kumar, P.Kumar: A Synchronous Checkpoiting Protocol for Mobile Distributed System: Probabilistic Approach, International Journal of Information and Computer Security 1(3)(2007), 298-314.
14. S. Neogy, A. Sinha, P.K. Das : A Checkpointing Protocol for Distributed System Processes, TENCON 2004. 2004 IEEE Regions 10 congerence vol.B. no.2,pp 553-556, November 2004, Thailand.
15. Parveen Kumar: A low-cost hybrid coordinated checkpointing protocol for mobile distributed systems, Journal of Mobile Information Systems, vol 4, Number 1, 2008.
16. S.K.Gupta, R.K.Chauhan and Parveen Kumar: An Efficient Snapshot Collection Protocol for Deterministic Mobile Distributed System, Int' J' of Computer Science and Engineering Systems, Vol. 2, No. 2, April 2008.
17. Guohui Li, LihChyun Shu: A Low-Latency checkpointing Scheme for Mobile Computing Systems.