Volume 4, Issue 4, April 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Cluster based Checkpointing Approach for Mobile Distributed Systems

Pradeep Kumar
Research Scholar,
Dept. of Computer Science,
Mewar Univ., Chittorgarh, India

Parveen Kumar
Deptt. of CSE,
Bhart Inst of Engg. & Tech.,
Meerut(UP), India

Surender Kumar
Deptt. of IT Engg.,
HCTM Technical Campus,
Kaithal(HRY), India

Abstract--- Mobile Hosts (MHs) are increasingly becoming common in distributed systems due to their availability, cost, and mobile connectivity but large amount of checkpointed data are not stored on local MHs memory. The limited stable storage available in mobile environments can make traditional fault tolerance techniques unsuitable. Since storage on a mobile host is not considered stable, most protocols designed for these environments save the checkpoints on the base stations. Previous approaches have assumed that the base station always has sufficient space for storing checkpoints. If there is not enough storage available, checkpointing may need to be aborted. An adaptive fault tolerance protocol is described in this paper for manages storage effectively. In cluster based architecture, whole network is divided into several cells and each cell has a Base Station(BS). Cluster has more than one BS it. BSs are the nodes that are given the responsibility for routing the messages within the cell and performing the data aggregation. The communication between two adjacent cells are conducted through the Base Station.

Keyword--- Fault tolerance, Base Station, Mobile Host, Mobile Distributed computing, cluster.

I. INTRODUCTION

With recent advances in mobile technology and mobile devices, mobile computing has become an important part of our life. People are using wireless networks for their day-to-day work, be it making a phone call or to download news or to see and listen or only listen to their favourite song from various multimedia servers with the help of various devices such as mobile phones, PDAs or a laptop. More services are in the offering in near future. The desire to be connected anytime, anywhere, anyhow has led to the development of wireless networks, opening new vista of research in pervasive and ubiquitous computing. This emerging field of mobile and nomadic computing requires a highly failure free environment to effectively manage the communication among the peers.

A mobile distributed system computing system is a distributed system where some of processes are running on mobile hosts (MHs) [1]. 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. 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 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. 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. A static node that has no support to MH can be considered as an MSS with no MH. Critically applications are required to execute fault-tolerantly on such system. 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 i.e. there exists a FIFO channel from an MH to its local MSS and another FIFO channel from the MSS to the MH .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. Message communication from an MH MH1 to another MH MH2 occurs as follows .MH1 first sends the message to its local MSS MSS1 using wireless link .MSS1 forwards it to MSS2, the local MSS of MH2 via a fixed network. Fig. 1 describes about The Mobile Distributed System [1]. Wireless backbone architecture can be used to support efficient communications between nodes [1], [2], [3], [5]. To support backbone architecture, the BSs should be a part of the backbone and the fewer the number of backbone nodes the better. Fewer nodes in the backbone can reduce the quality of messages exchanged by backbone nodes [3], [4]. In this paper, we propose a fault tolerance scheme for clustering routing protocol as a method of improving reliability. BS sends routing and collected data information to its cluster which is very nearest to MSS, which periodically save the state of BS. If a BS fails or some

fault is detected, then cluster head detects the BS failure and responsibility of BS is assigned to any other BS among the cluster. Using fault tolerance the cell can quickly recover from a transient fault of BS.

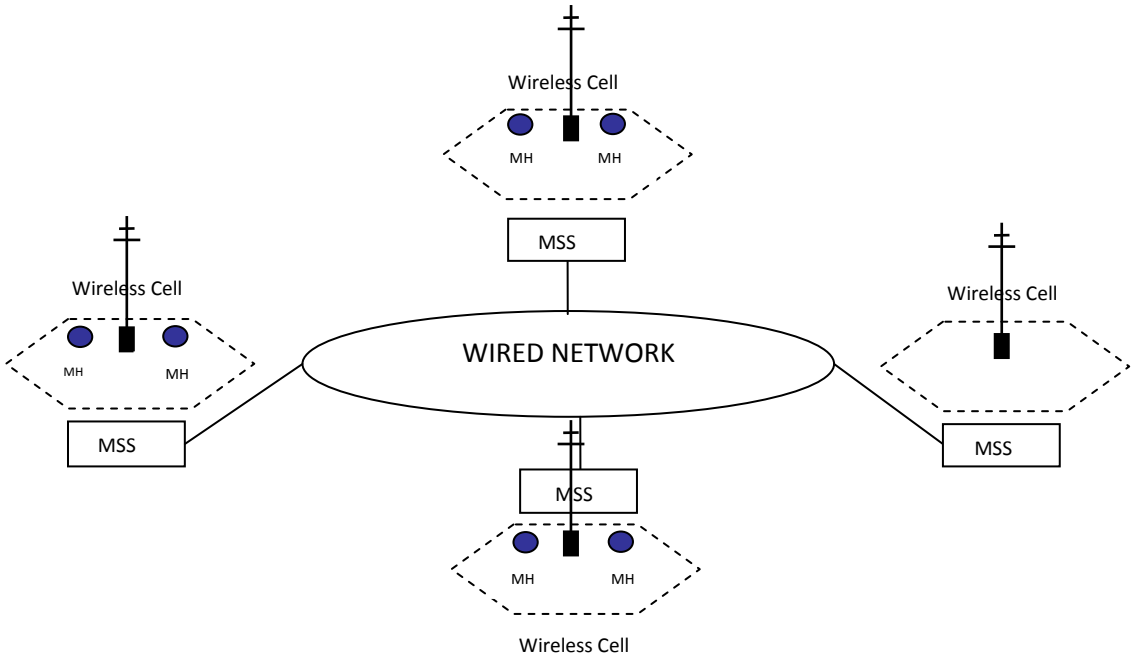

Fig. 1 Working Block Diagram of a Mobile Distributed System

Related work and problem formulation in Section II and system model in section III. The proposed fault tolerance scheme is formulated in Section IV, Section V shows the performance of our algorithm and finally, Section VI concludes the paper.

II. RELATED WORK AND PROBLEM FORMULATION

A. Related work

In this section we briefly introduce prior studies related to our work. In [6], the authors proposed the concept of in-network fault tolerance for achieving enhanced network dependability and performance. In that scheme, the sink node periodically checkpoints its state and saves it in the memory of one or more sensor nodes, so called checkpoint sensors. When a sink node (S_1) fails or reaches an energy level below its threshold, another sensor node will be selected to operate as the new sink node (S_2). After applying this approach m times, the sink will be located in a sensor denoted by Sm. If the sink is located on S_m, then S_{m-1} is the checkpoint sensor and the path between S1 and S_m is the checkpoint path. When a sink node (S_m) fails, S_{m-1} detects the failure and becomes the sink instead; it iteratively operates in this sequence through the checkpoint path. This scheme is simple to implement, but energy consumption and reliability vary according to the position of the sink node.

In [7], a cell takes two types of checkpoints – processes inside the cell take synchronous checkpoints and a cell takes a communication induced checkpoint whenever it receives an inter-cell application message. Each cell maintains a sequence number (SN). SN is incremented each time a cell level message is committed.

In [8], authors proposed a simple non-blocking roll-forward fault tolerance/recovery mechanism for cell federation. The main feature of their algorithm is that a process receiving a message does not need to worry whether the received message may become orphan or not. It is the responsibility of the sender of the message to make it non-orphan.

In [18], the authors proposed an integrated independent and coordinated fault tolerance schemes for the applications running in hybrid distributed environments. They stated that independent checkpoint subsystem takes a new coordinated checkpoint set if it sends an inter-cell application message. Also a process pi of independent fault tolerance subsystem takes a new independent checkpoint before processing an already received inter-cell application message, if pi has sent any intra-cell application message after taking its last checkpoint. In [9] and [10] the authors have proposed non-blocking coordinated fault tolerance algorithms that require minimum number of processes to take checkpoints at any instant of time.

A good fault tolerance protocol for mobile distributed systems should have low overheads on MHs and wireless channels and should avoid awakening of MHs in doze mode operation. The disconnection of MHs should not lead to infinite wait state. The algorithm should be non-intrusive and should force minimum number of processes to take their

local checkpoints [11]. In minimum-process coordinated fault tolerance algorithms, some blocking of the processes takes place [12], [13], or some useless checkpoints are taken [9], [10], [14].

Cao and Singhal [9] achieved non-intrusiveness in the minimum-process algorithm by introducing the concept of mutable checkpoints. The number of useless checkpoints in [9] may be exceedingly high in some situations [14]. Kumar et. al [14] and Kumar et. al [10] reduced the height of the fault tolerance tree and the number of useless checkpoints by keeping non-intrusiveness intact, at the extra cost of maintaining and collecting dependency vectors, computing the minimum set and broadcasting the same on the static network along with the checkpoint request.

Higaki and Takizawa [15] proposed a hybrid fault tolerance protocol where the mobile stations take checkpoints asynchronously and fixed ones synchronously. Kumar and Kumar [17] proposed a minimum-process coordinated fault tolerance algorithm where the number of useless checkpoints and blocking are reduced by using a probabilistic approach. A process takes its mutable checkpoint only if the probability that it will get the checkpoint request in the current initiation is high. To balance the fault tolerance overhead and the loss of computation on recovery, P Kumar [16] proposed a hybrid-coordinated fault tolerance protocol for mobile distributed systems, where an all-process checkpoint is taken after executing minimum-process fault tolerance algorithm for a certain number of times.

B. Problem Formulation

The limited stable storage available in mobile environments can make traditional fault tolerance techniques unsuitable. Since storage on a mobile host is not considered stable, most protocols designed for these environments save the checkpoints on the base stations. Previous approaches have assumed that the base station always has sufficient space for storing checkpoints. If there is not enough storage available, checkpointing may need to be aborted. An adaptive storage fault tolerance protocol is described in this paper for manages storage effectively for base stations. In cluster based architecture, whole network is divided into several cells and in each cell there is a Base Station (BS) and more than one cell form a cluster.

III. SYSTEM MODEL

In mobile distributed network each cells CLs, has a BS and possibly some Mobile Host (MH)s. BS acts as a local coordinator of transmissions within the cell. Each cell is represented by the ID of its BS. For example, Fig. 2 shows a cell based distributed mobile computing systems and there are four cells CL1, CL2, CL3 and CL4. A MH can communicate with other MHs in different cell or in the same cell only through its own BS. This architecture is characterized by two types of messages – inter-cell messages and intra-cell message. The main aim of approach is to efficiently maintain energy consumption of nodes by involving them in multi-hop communication within a particular cell and by performing data aggregation in order to decrease the number of messages transmitted to MSS. Since, normal nodes only communicate with their BS, which in turn, aggregates the collected information and sends it to its cluster heads and cluster head forwarded it to the MSS. In this scheme, BS failures are more critical than those normal nodes. When a BS fails, re-election of BS is performed within the cluster. Such a recovery scheme is a time and energy consuming process. Therefore, to improve the quality and reliability of mobile distributed networks, a fault tolerant mechanism is needed for such BSs.

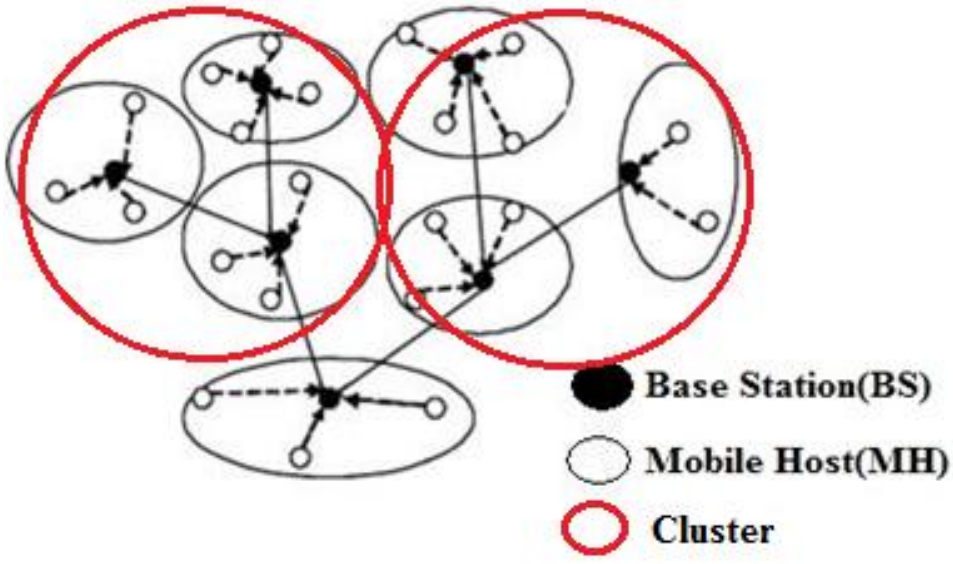

Fig. 2 System Model of Proposed System

We have proposed a non-blocking coordinated fault tolerance algorithm in which MHs take a tentative checkpoint and then on receiving a commit message from the initiator, the MHs convert their tentative checkpoint into permanent. Also whenever a MH is busy, the process takes a checkpoint after completing the current procedure. The proposed algorithm requires fewer control messages and hence fewer number of interrupts. Also, our algorithm requires only minimum number of MHs in a cell to take checkpoints; it makes our algorithm suitable for mobile distributed networks.

IV. FAULT TOLERANCE ALGORITHM

Mobile distributed network have any predefined set up or structure. Nodes may always be moving and there may be frequent link failures. When a node fails, all other nodes learn the failure in finite time. We assume that the fault tolerance algorithm operates both intra-cluster, inter-cluster and in the cell. Nodes are referred to as process. Consider a cell having a set of n nodes { N1, N2……N_i} involved in the execution of the algorithm. Each node N_i maintains a dependency vector $DVector_i$ of size n which is initially empty and an entry $DVector_i[j]$ is set to 1 when N_i receives since its last checkpoint at least one message from N_j. It is reset to 0 again when Node N_i takes a checkpoint. Each node N_i maintains a checkpoint sequence number csn_i. This csn_i actually represents the current fault tolerance interval of node N_i. The ith checkpoint interval of a process denotes all the computation performed between its i^{th} and $(i+1)^{th}$ checkpoint, including the i^{th} checkpoint but not the $(i+1)^{th}$ checkpoint. The csn_i is initially set to 1 and is incremented when node N_i takes a checkpoint. In this approach, we assume that only one node can initiate the chekpointing algorithm. This node is call as initiator node. We define that process N_k is dependent on another process N_r, if process N_r since its last checkpoint has received at least one application message from process N_k. In our proposed scheme, we assume primary and secondary checkpoint request exchanges between BS and rest n-1 Mobile Host(MH)s. A permanent checkpoint request is denoted by $R_i(i=csni)$ where i is the current checkpoint sequence number of BS that initiates the fault tolerance algorithm. It is sent by the initiator process $N_j$to all its dependent nodes asking them to take their respective checkpoints. A tentative checkpoint request denoted by R_{si} is sent from process N_m to process N_n which is dependent on N_m to take a checkpoint R_{si} means to its receiver process that its the current checkpoint sequence number of the sender process. When P_i send sm to P_j, P_i piggybacks c- $state_i$, own_csn_i along with m. c_state_i A flag. Set to '1' on the receipt of the minimum set. Set to '0' on receiving *commit* or *abort. own*_csn is the csn of P_i at the time of sending m. Consider a distributed system with n processes. P0, P1, P2, P3 and P4. Each process Pi in the system maintains ddvi vector of size n which is initially set to zero and an entry in ddvi[j]=1 when Pi receives a commutation message during current checkpoint interval(CI). A process Pi sends a computation message m with ddvi [] to Pj. Pj updates its own ddvj after receiving m as follows: ddvj[k] = 1 or ddvi[k] m.ddv[k], $1 \le k \le n$ and $\vee$ is the bitwise inclusive OR operator. Hence if Pi depends on Pk before sending m, Pj also become dependent transitively on Pk after receiving m. Also each process Pi piggybacks its icsn with only every first outgoing computation message to process Pj after taking checkpoint.

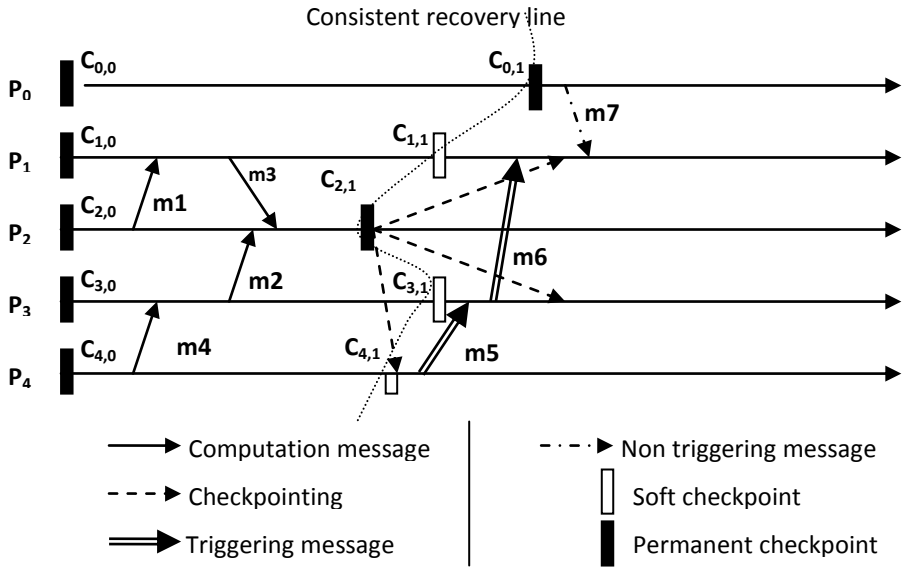

Fig. 3 Working example of proposed protocol

Each process Pi 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 P1 to P5 in a distributed system fig. 3. We assume that process P2 initiates the checkpointing process. The ddv[] vector are maintained as follows: When process P4 sends a message m4 to P3, it appends ddv4[00001] to the message. When P3 receives message m4, it extracts the boolean vector ddv4 [] from the message and updates its ddv3 by taking the OR of ddv3 [00010] and ddv4 [00001].Now the updated ddv3 become [00011] and sends this updated vector by appending with the message m2. Similarly by updating ddv2 [] on the receipt of message m2 at process, ddv2 [] become [00111]. In same way P1 updates its ddv1 [01000] to [011000] on receiving the message m1 from process P2 and sends this updated

ddv2 [] vector with message m3. At last after receiving the message m3 from P1 the ddv2 become [011111] which shows that process P1,P3, P4 are directly or transitively dependent on P2. We make the observations related to our algorithm (a) Process P1, P3 and P4 are the part of minimum set and receives the checkpointing request from the initiator process P2 (b) Checkpoint C0,1 and C2,1 are the permanent checkpoint and stored at stable storage of MSS and non-triggering message are sent after these (c) Checkpoints C4,1, C3,1, and C1,1 are the soft checkpoint, and triggering computation message are sent after these. So at the time of initiation, P2 take its own permanent checkpoint and sends checkpoint request to P1, P3 and P4, increment its csn C2,0 to C2,1 and wait for reply or acknowledgement.

When P2's request reaches P4, it takes soft checkpoint C4,1 and sends triggering computation message(as it is sent after taking the soft checkpoint) m5 to P3. Process P3 receives triggering computation message before it receives the checkpoint request message from the initiator, it first compares (csn3 [Pid] ≠ m.Pid) and concludes that P3 does not participate in current checkpointing initiation and asked to participate. Secondly it takes the soft checkpoint C3,1 before processing the message m3 as the csn3[4] which is 1, smaller than P4's current csn4=2 received with message m5.

Similarly process P1 takes its soft checkpoint C1,1 after receiving the triggering computation message from P3. Later when P1 and P3 receive the checkpoint request from the initiator P2, these processes ignore the checkpoint request as they have already participated in current initiations. Process P0 which are not the part of minimum set, takes its permanent checkpoint C0,1 independently and sends the non-triggering computation message m6 to process P1 by appending C0,1. After receiving non-triggering computation message m6 process P1 observe that C0,1 is not soft checkpoint; it only update the csn1[0] to 1, receives the message and conclude that new checkpoint is not necessary.

At last when initiator processes P2 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 P2 sends the ABORT message to all the participating process. After receiving the ABORT message, processes discard their soft checkpoint taken related to current CI.

Algorithm

We define the pseudo code here.

(a) *Action on the initiator P_j:*

Send checkpoint request to its Cluster head CH

(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) Upon receiving COMMIT message

- Make soft checkpoint permanent;
- Increment own csn;

(g) Upon receiving ABORT message

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

Consider the pseudo code for any node N_j. Node N_j makes sure that all processes from which it has received messages also take checkpoints so that there are no orphan messages that it has received. Also, the node N_j first takes its checkpoint if needed, then processes the received piggybacked application message. Thus, such messages cannot be an orphan. Hence, algorithm generates a consistent global state.

V. PERFOMANCE ANALYSIS

We compare our algorithm with [7],[9],[10] and[18]. Comparative table 1 as shown below:

Table 1 : - Comparison with the related work

Parameters	**[7]**	**[18]**	**[9]**	**[10]**	**Our algorithm**
Non-blocking	Yes	Yes	Yes	Yes	Yes
Minimum Process	No	No	Yes	Yes	Yes
Supports MANET's	Yes	Yes	No	No	Yes
Number of checkpoints	Less	More	Less	Less	Less
No. of control messages	More	More	Less	Less	Less

VI. CONCLUSION

In this paper, we have proposed a minimum process and non-blocking fault tolerance scheme for clustering routing protocols. The protocol proposed by us is non-blocking and suitable for mobile environments. It produces a consistent set of checkpoints. The algorithm makes sure that only minimum numbers of nodes in the cluster are required to take checkpoints; it uses very few control messages. Performance analysis shows that our algorithm outperforms the existing related works and is a novel idea in the field. And finally, it reduces the energy consumption and recovery latency when a BS fails.

REFERENCES

[1] D.J. Baker and A. Ephremides, "The Architectural Organisation of a Mobile Radio Network via a Distributed algorithm", IEEE Trans. Commun., vol. 29, no. 11, pp 1694-1701, Nov., 1981

[2] D.J. Baker, A. Ephremides and J.A. Flynn "The design and Simulation of a Mobile Radio Network with Distributed Control", IEEE J. sel. Areas Commun.., pp 226-237, 1984

[3] B.Das, R. Sivakumar and V. Bharghavan, "Routing in Ad-hoc networks using a Spine",Proc. Sixth International Conference, 1997.

[4] B.Das, R. Sivakumar and V. Bharghavan, "Routing in Ad-hoc networks using Minimum connected Dominating Sets",Proc. IEEE International Conference, 1997.

[5] M.Gerla, G. Pei, and S.J. Lee, "Wireless Mobile Ad-hoc Network Routing", Proc. IEEE/ACM FOCUS'99, 1999.

[6] Iman, S.; Adnan, A.; Mohamed, E. In-network fault tolerance in networked sensor systems. In Proceedings of the Workshop on DEPENDABILITY ISSUES in WIRELESs Ad Hoc Networks and Sensor Networks, Los Angeles, CA, USA, 26–27 September 2006; pp. 47-54.

[7] S.Monnet, C.Morin, R. Badrinath, " Hybrid fault tolerance for parallel applicatuions in cluster federation", In 4th IEEE/ACM International symposium on cluster computing and the Grid, Chicago, USA, pp 773-782, April, 2004.

[8] B. Gupta, S.Rahimi and R. Ahmad, "A New Roll-Forward Chekpointing /Recovery Mechanism for cluster federation", International Journal of Computer Science and Network Security, Vol. 6, No. 11, Nov., 2006.

[9] G.Cao and M.Singhal, "Mutable checkpoints : A new fault tolerance approach for mobile computing systems", IEEE Transactions on parallel and Distributed Systems, 12(2), 157-172, Feb., 2001

[10] P.Kumar, L.Kumar , R.K. Chauhan and V.K. Gupta, "A Non-intrusive Minimum process Synchronous Fault tolerance Protocol for Mobile Distributed Systems", ICPWC 2005, IEEE International Conference on Personal Wireless Communications, 491-495, New Delhi, Jan., 2005.

[11] Prakash R. and Singhal M., "Low-Cost Fault tolerance and Failure Recovery in Mobile Computing Systems," IEEE Transaction On Parallel and Distributed Systems, vol. 7, no. 10, pp. 1035-1048, October1996.

[12] Cao G. and Singhal M., "On the Impossibility of Min-process Non-blocking Fault tolerance and an Efficient Fault tolerance Algorithm for Mobile Computing Systems," Proceedings of International Conference on Parallel Processing, pp. 37-44, August 1998

[13] Koo R. and Toueg S., "Fault tolerance and Roll-Back Recovery for Distributed Systems," IEEE Trans. on Software Engineering, vol. 13, no. 1, pp. 23-31, January 1987.

[14] L. Kumar, M. Misra, R.C. Joshi, "Low overhead optimal fault tolerance for mobile distributed systems" Proceedings. 19th IEEE International Conference on Data Engineering, pp 686 – 88, 2003.

[15] Higaki H. and Takizawa M., "Checkpoint-recovery Protocol for Reliable Mobile Systems," Trans. of Information processing Japan, vol. 40, no.1, pp. 236-244, Jan. 1999.

[16] Parveen Kumar, "A Low-Cost Hybrid Coordinated Fault tolerance Protocol for mobile distributed systems", To appear in Mobile Information Systems.

[17] Lalit Kumar Awasthi, P.Kumar, "A Synchronous Fault tolerance Protocol for Mobile Distributed Systems: Probabilistic Approach" International Journal of Information and Computer Security, Vol.1, No.3 pp 298-314.

[18] J.Cao, Y.Chen, K.Zhang and Y.He, "Fault tolerance in Hybrid Distributed Systems", Proc. Of the 7th International symposium on parallel architectures, algorithms and Networks (ISPAN'04), pp 136-141, Hong Kong, China, May, 2004.

