Proxy MSS based Synchronous Checkpointing Approach for Mobile Distributed Systems

Pradeep Kumar Sharma
Research Scholar, Dept. of Computer Science, Mewar Univ., Chittorgarh(Raj.)

Parveen Kumar
Professor, Bharat Institute of Engineering & Technology, Meerut (UP)

Surender Jangra
Associate Professor, Deptt. of IT Engg., HCTM, Kaithal(Haryana)

Abstract: Mobile Distributed systems (MDSs) are made up of Mobile host (MH), Base Station (BS) and Mobile Support Station (MSS). Among which MSSs play a key role in mobile environment. This paper presents a low overhead Proxy MSS based framework to handle the fault in mobile distributed Systems. In the proposed scheme, one MSS lot of proxy MSS works as per the workload. One proxy MSS handles the specific group of BS. It is also suggested that all checkpointing data and their related overheads are forwarded to the proxy MSSs and as a result the workload of Mobile Hosts (MHs) will reduce substantially and battery power can be saved. Moreover, all the coordinated message and checkpoint requests will be decreased. The proposed non-blocking proxy based synchronous checkpointing approach has a simple data structure and forces minimum numbers of process to take checkpoint . At the end of the paper simulation results show the comparative analysis in different perspectives. Thus, in view of its unique features, the proposed scheme would be efficient and suitable for mobile distributed systems.
Keword: Checkpointing, Mobile Host (MH), Mobile Support System (MSS), Proxy MSS.

I. INTRODUCTION

To make the distributed application robust and recoverable against various types of failure, lot of checkpointing algorithms developed in the past years [2], [3], [4], [6-8]. However, little attention has been devoted to checkpointing algorithms for MDSs[Figure 1]. Now a day's wireless networks, and mobile devices become pervasive, it is necessary to extend the capability of checkpointing to wireless and mobile environment. The MHs have several new characteristics like host mobility, limited battery power and lack of stable storage. In addition, wireless connections is more unreliable in term of frequently disconnections, limited bandwidth, limited geographical area compared to wired connection. Due to the above unique characteristics of mobile environments, it is not appropriate to directly apply checkpointing and recovery protocols designed for distributed systems to MDSs [1], [5]. So any checkpointing approach for fault tolerant in mobile environment should consider these distinguishing features in their application. Hence the recent research also considered for mobile computers [1], [5], [9-16].

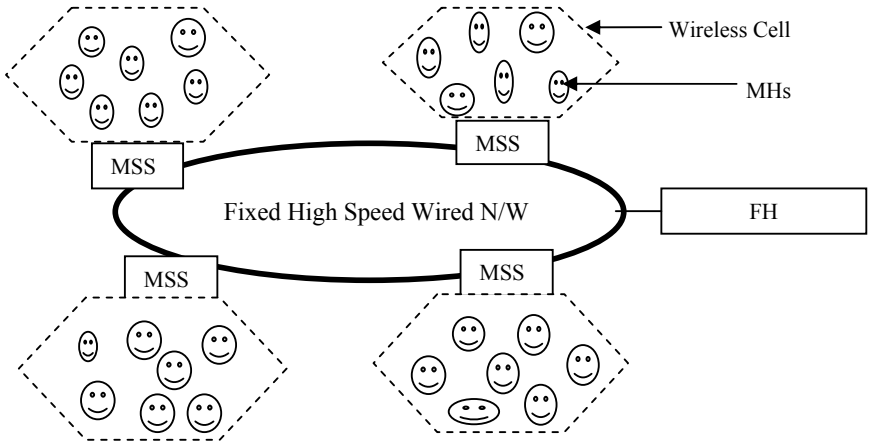

Figure 1 Working Block Diagram Mobile Distributed System (MDS)

Mostly checkpointing algorithms for distributed and mobile systems follow two-phase [1], [2], [4]. In the first phase initiator process forwards tentative checkpoint request to all its dependent nodes and when it knows that all the participated processes takes tentative checkpoints successfully, then it forwards the permanent checkpoint request in the second phase. Both the tentative and permanent checkpoints are stored on the stable storage of the

MSS. A permanent checkpoint is a local checkpoint at a process and is a part of a CGC. A tentative checkpoint is a temporary checkpoint that is made a permanent checkpoint on the successful termination of the checkpoint algorithm. In case of a single failure, all the participating processes roll back only to their permanent checkpoints for recovery.

II. RELATED WORKS AND PROBLEM FORMULATION

MDSs environments require efficient use of the limited bandwidth and limited resources of portable devices, such as battery power and memory etc. Mobile devices have low computation power, memory and wireless bandwidth between MHs and MSS. Hence, transferring unnecessary checkpoints and rollbacks may waste a large amount of computation power, bandwidth and energy. All the above traditional two phase checkpointing is an expensive fault tolerance method for two reasons: (i) it may involve unnecessary node rollbacks (ii) it requires large number of control messages. As a result it affects the bandwidth and power negatively. Existence of mobile nodes in a DS, introduces new issues likes mobility, disconnection, finite power source and lack of stable storage. So compared to traditional distributed environment, mobile networks are typically slow with low bandwidth and throughput.

Cao and Singhal [5] introduce a mutable checkpoint approach for mobile systems by assuming that MHs have the sufficient memory to store the checkpoint temporarily. These mutable checkpoints are neither a tentative checkpoints nor a permanent checkpoint and need not to be saved on stable storage. These mutable checkpoints can be saved anywhere, e.g., the main memory or local disk of MHs. Taking such mutable checkpoints avoids the overhead of transferring large amount checkpoint data to the stable storage of MSS over the wireless network and during failure processes rollback locally to previous consistent committed state by discarding the mutable checkpoint.

Figure 2, shows the inconsistency exists in Cao and Singhal's algorithm [5]. 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.

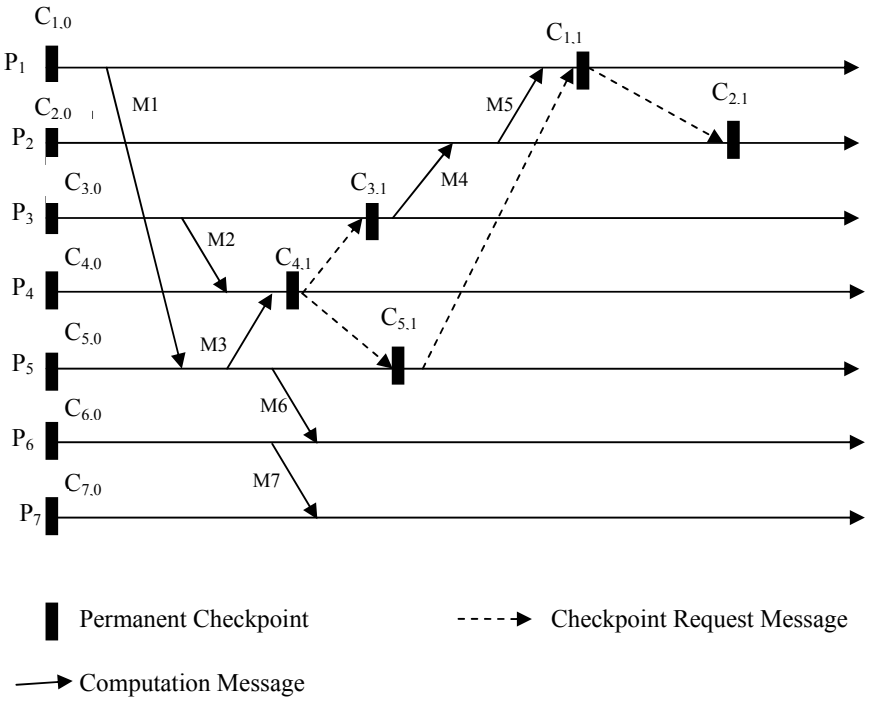

Figure 2 An example showing inconsistency in algorithm [5]

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 sent 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. Although this scheme produces CGS with little overhead but it also fails to overcome the storage

overhead on mobile devices.

In this paper, we have designed a proxy based fault tolerance architecture and algorithm for MDSs which ensures consistency, accounts for mobility, provides fault tolerance and recovery in mobile environment with minimum overhead.

III. PROPOSED PROXY BASED ARCHITECTURE

MDS is a distributed system where some part of the computation are executed on large number of mobile hosts(MHs) and some part on few static hosts (SHs). MHs are more vulnerable than SHs to both communication and failures, as they are more prone to theft, loss, disconnection and accidental damage since user carry them around with them. In addition, there is a wireless connection between MHs and MSS which is more unreliable and failure prone than wired. Due to above reasons stable storage on MHs are not considering stable and MHs transfers their data on stable storage of their local MSS. To take into account the vulnerability of the MHs in to MDSs we design new proxy based frameworks to handle fault.

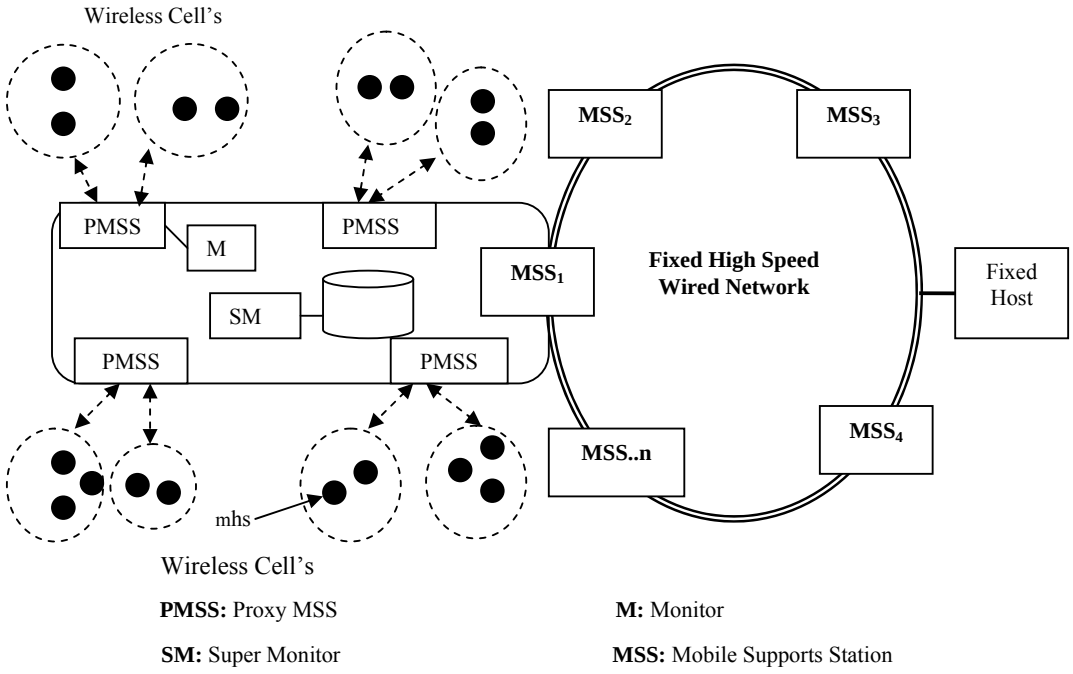

Figure 3 Proxy based architecture

In the architecture shown in Figure 3, MHs are executed on mobile devices which are connected over wireless link using a network protocol that support host mobility with Proxy MSS (PMSS) and SHs are connected over wired link directly with the MSS using standard TCP/IP protocol. All messages exchange between one MH to another go through the PMSSs. PMSSs maintains process state on the behalf of the MHs. It keeps a copy of the MHs state in its memory and continuously updates the state as it receives messages to and from the client. After receiving the checkpoint request from the initiator node, PMSSs will just mark the latest checkpoint as tentative. It means there is no need to sends the checkpoint request to MHs over wireless link, as the tentative checkpoints contains the latest copies of states. However, if the copy of latest checkpoint expired from the stable storage of PMSSs, only then it asks to MH to send the latest checkpoint data again. We will use the term PMSS and MH interchangeably when there is no ambiguity. All the components are work as follow in the architecture:

Proxy MSS (PMSS): There is a single PMSS upon each wireless cell and all the communication between MH and MSS go through PMSS. First, MH sends message to PMSS, then PMSS forwards them to their destinations. On the receipt of checkpoint request from the initiator, PMSS will just mark the received snapshot as tentative(depending upon condition) and save it on its local memory, which will save the time to connect to MHs and receive the snapshot, as all the related information are available on it. It also notify to its local MSS about its willingness to take checkpoint by sending request message through wired link. As messages are transferred through PMSS, each PMSS maintains the following SEND and REPMSST table related to each MHs.

Table 1 SEND Table

Message Status	Message Seq. No.	Destination	Message ID

In the Table message have the following *status*:

0- Buffered message i.e., message received but not sent to destination
1- Forwarded message i.e., message sent to destination but not acknowledge
2- Processed message i.e., acknowledge for above sent message received
3- Archived message i.e., message sent during CI.

On the other hand RECPT table contains the following attributes where message status has values 0, 1 and 2. Here 0 represent that acknowledgement is sent, 1 for not acknowledged and 2 for, message received during previous CI.

Table 2 RECPT Table

ACK Status	Message Seq. No.	Source

If PMSS received a message from MSS but destination MH is currently disconnected, the message is labeled 0 as buffered and put them in to MBQ. When the disconnected MH reconnects, all the buffered message are forwarded in the same order as they are arrive at the PMSS. A *buffered message* that has been forwarded to its destination but whose acknowledge is not received at the PMSS is labeled 1 as *forwarded message* and it converted into label 2 as *processed message* after receiving the acknowledgement. Both buffered and forwarded message are available on the PMSS. In such way message status will modify at happening of any event.

Monitor (M): Monitors inspects the status of the systems state and keep the records of failure, disconnections and the location of mobile nodes. It also keeps the record of total number of message sent, received, and sequence number of each MHs and handle the lost message, orphan message and garbage collection activities.

Super monitor(SM): Super monitor helps in maintaining the global state of the system. It also maintains the information regarding failure, disconnection and location related information of all the MHs upon MSS level**.**

IV. SYSTEM MODEL

A mobile 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 and connected to Proxy MSS (PMSS) or Mobile Host Proxy (MHP) via a reliable network that supports mobility. An MH communicates with other nodes of system via PMSS and 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 PMSS. A cell is a geographical area around 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.

In our system model 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 and work on fail-stop model. Each process progresses at its own speed and messages are exchanged through reliable channels, whose transmission delays are finite but arbitrary. Communication between the MHs and the PMSS and between PMSSs and MSS are assumed to be lossless and FIFO. It also includes the processes of MHs, which have been disconnected from the PMSS but their checkpoint related information is still with this PMSS. MHs and PMSSs uses application level checkpoint sequence number to maintain the consistency and message sequence number to detect the lost or duplication of messages.

V. PROPOSED CHECKPOINTING ALGORITHM

1. On checkpoint initiation:

Let MH_q is an MH_{in} and initiates the checkpointing process, it sends the checkpoint initiation request to its local PMSS and resumes its working. If PMSS is already participated in any GC recording then it discards the request and informs to the MH_q. On the other hand, on receipt the checkpoint initiation request it takes a tentative checkpoint on its own local storage, increments its csn, set weight to 1, set its PMSS_state to 1, set trigger set(pid_q, $csn_q[q]$), set $minset_q[]$ and sends checkpointing request to all the PMSSs such that $minset_q[i]=1$ and where $i \neq q$.

2. On receiving checkpointing request:

Upon receipt of the checkpoint request with trigger, minset[] and weight from $PMSS_q$, $PMSS_p$ will compare recv_csn with its old_csn, that is if and only if $old\_csn_p \leq$ recv_csn and deciding to take checkpoint. After receiving the request it reply negatively or positively to the initiator nodes and also forwarded this message to its dependent nodes which are not a part of minset.

3. Termination of algorithm:

At $PMSS_q$, when it is conforms that all concerned processes takes checkpoints successfully means weight become 1, it sends commit message to all PMSSs to convert their checkpoints into permanent one. In another case if time out occurs or it receives negative acknowledgement from any MH, then it forwards *abort* message.

4. On receiving commit or abort message:

All PMSSs converts their tentative checkpoints into permanent one after receiving the commit message. If it receives the abort message, CPs will discard the tentative checkpoints from its personal storage memory but data is available on PMSSs. Hence, our proxy based approach does not awake the MHs during checkpointing and requires very less control message over wireless network.

5. On receiving and sending computation message during checkpointing:

When a process P_i runs on MH_i sends a computation message to process P_j which run on MH_j, it piggybacks his csn, trigger, and minset with the message. This computation message proceeds through the PMSS both at sending and receiving end i.e. MH_i first sends message to its PMSS which is $PMSS_i$, then $PMSS_i$ sends this message to $PMSS_j$, and at last $PMSS_j$ transfer this message to its destination MH which is MH_j. On receiving end following actions are taken by $PMSS_j$:

a) if ($old\_csn_j[i] >= m.csn_i[i]$): it means both the processes take 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 gets the checkpointing request from the initiator).

(ii) if (($P_i \notin minset[]$) $\wedge$ (Bitwise logical AND of $sendv_i[]$ $\wedge$ 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 (($P_i \notin$ minset[]) $\wedge$ (Bitwise logical AND of $sendv_i[]$ $\wedge$ minset[] is all zero)) : in such case there is a probability that a process do not get any checkpoint request; process buffers the message.

VI. **WORKING EXAMPLE**

We explain our checkpointing algorithm with the help of an example. Consider the distributed system as shown in Figure 4. Note that when a computation message is sent after taking the checkpoint it piggybacked with minset[]. The entire computation message is piggybacked with the csn and dependency vector. Assume that process P_4 initiate checkpointing process in Figure 4. First process P_4 takes its 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 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 attaching 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 $C_{4,1}$.

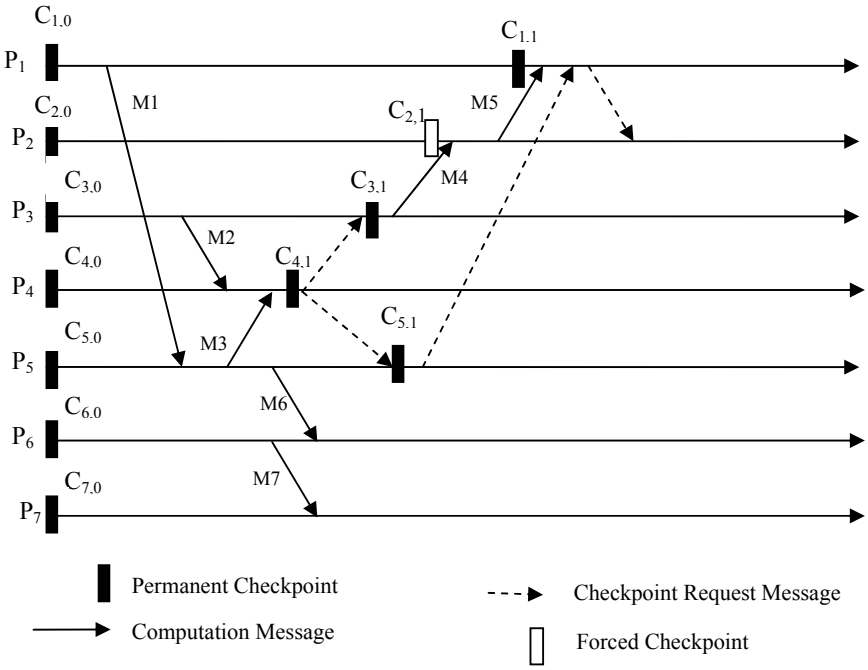

Figure 4 An example of proposed approach

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{ P_1, P_3, P_4, P_5} 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 receives 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.

VII. PERFORMANCE ANALYSIS

Before analyzing the algorithm, we understand the factors that affect the recovery in distributed and mobile systems. The following factors affect the recovery [17]:

- *Failure rate of MHs:* Due to the wireless connectivity with network and limited battery power MHs are more prone to failure. If failures are more frequent, then transaction has to rollback every failure, which increases the total execution time.
- *Memory constraints:* Storing data on the PMSS requires lots of memory space on the PMSS. Hence, it is requires to adopt some garbage collection approaches for PMSSs.
- *Wireless connectivity:* Due to weak wireless link between MH and PMSS, only essential write event should be transferred over wireless link.
- *Recovery time:* The recovery time depends upon the recovery scheme and methods used for checkpointing algorithm.

In our approach all the log information are stored in Proxy MSS (PMSS). When a MH moves within the same cell, no log information is transferred, as PMSS handles all the data of the cell. In such way handoff, disconnection, and failure cost can be reduced. When a MH moves in another cell, then it carries the id number of previous PMSS and its own id, for the purpose of registration with other PMSS. When a MH register with

other PMSS, then the previous PMSS sends all the log information to the new PMSS over wired link. In [89] the authors presented the following relationship between recovery time, failure rate, handoff threshold, and recovery probability.

- *Recovery rate vs. failure rate:* Both the failure rate and recovery time is contradicted to each other. Failure recovery affects the checkpoint information as higher the failure rate, fewer the checkpoint information are transferred through wireless link.
- *Recovery time vs. Handoff threshold:* handoff threshold time also affects to the recovery time as lower value of the handoff threshold, requires multiple handoff which has the fewer checkpoint information, and takes less time during recovery. Hence, recovery time increase the handoff threshold.
- *Recovery probability vs. failure rate:* As a higher failure rate indicates lesser computation, so lesser log entries are created due to checkpointing. Hence, recovery probability increases as the failure rate increases.

In order to evaluate proxy MSS based approach, we simulate a message passing based mobile environment with a number of MSSs and MHs which are randomly placed in these MSSs initially. In fact, an MH sends a message to its supporting MSS in order to deliver it to another MH. This message is routed among the system's MSSs in order to reach the supporting MSS of the destination MH, and finally it reaches the destination MH through its supporting MSS. We consider random delivery delays for each of these message transmissions. An MH triggers a checkpointing process randomly so it sends the related request to its supporting MSS, and other relevant reactions based on sending and receiving batch checkpoint requests take place in response to this checkpoint initiation according to the implementation of our proposed algorithm in the simulation.

We compared the results in figure 5,6 and 7 of our proposed algorithm with Cao and Singhal's algorithm and MSS based protocols. Moreover, in the proposed approach is orphan message free. The advantages mentioned indicate that the proposed checkpointing scheme presents a more practical perspective of tolerating mobile distributed systems against faults, and is also efficient and suitable for such systems.

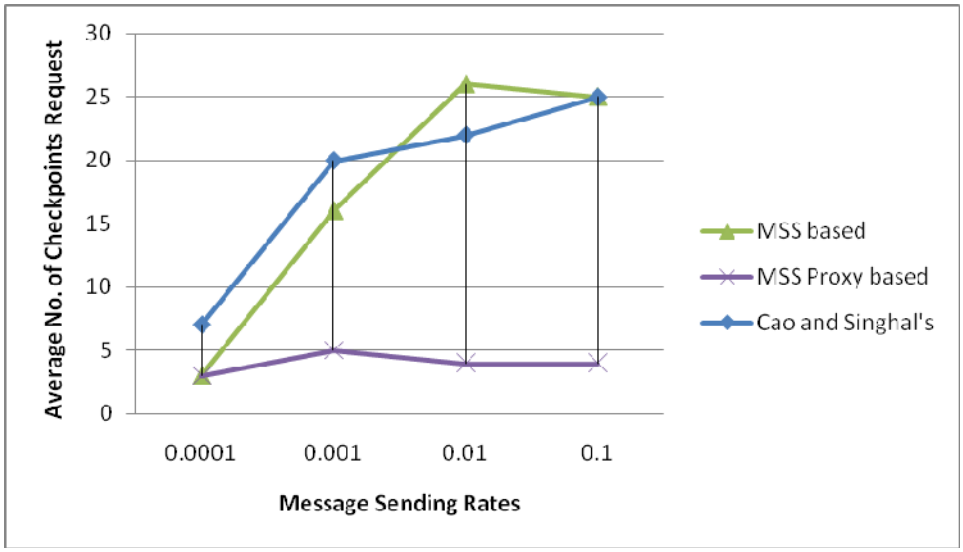

Figure 5 Number of Checkpoint Requests

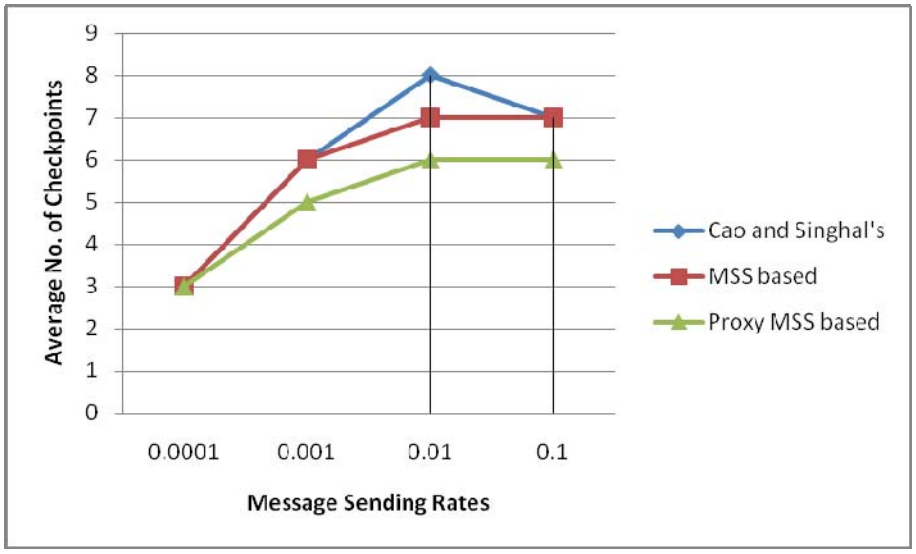

Figure 6 Number of Checkpoints

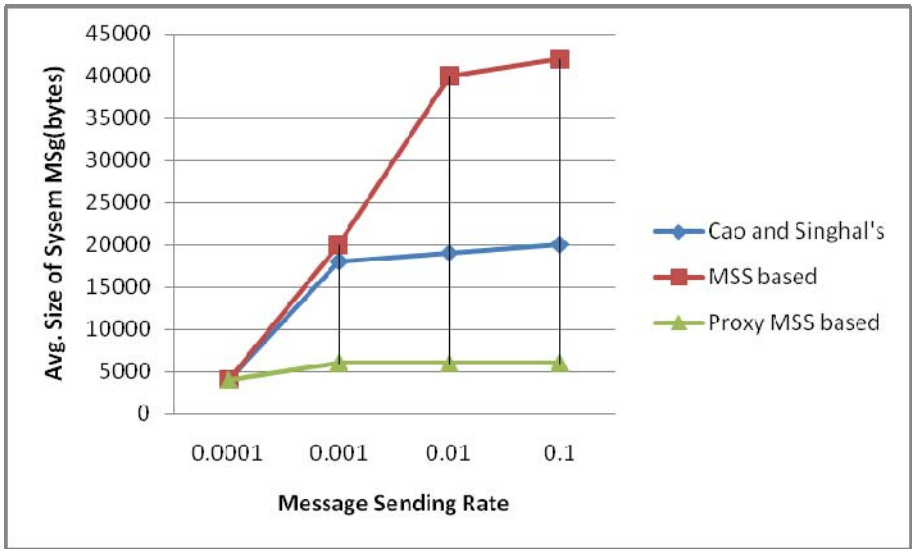

Figure 7 Size of System Message (bytes)

VIII. CONCLUSION

The existence of mobile nodes in a distributed system introduces new issues and makes the traditional algorithm inadequate for MDSs. These issues are host mobility, frequently disconnection, weak wireless connectivity, limited power of hand-held devices, lack of stable storage on MHs hand-held devices etc. In this paper, we presented a proxy MSS based checkpointing scheme which may be very useful for MDSs as it reduces the recovery time and overheads on large scale. In this approach checkpoints are stored on resource rich Proxy MSS (PMSS) for saving the processor cycle, memory and power of wireless hand-held devices. MHs need not send any extra coordination messages through the wireless network and so battery power is also saved. Since PMSS hides frequent client disconnection, handoff, failure from the MSS, fewer reconnect requests are sent to MSS. On the other hand, since our approach is based on non-blocking coordinated checkpointing approaches, the system would take a minimum number of checkpoints which makes it suitable of mobile and hand held devices.

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

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

[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] R K Chauhan, Parveen Kumar Lalit Kumar, "*A coordinated checkpointing protocol for mobile computing systems*", Int'l Journal of information and computing science, Accepted for Publication, Vol. 9, No. 1, 2006.

[7] Chauhan R.K., Kumar P. and Kumar L., "*Non-intrusive Coordinated Checkpointing Protocol for Mobile Computing Systems: A Critical Survey*", ACCST Journal of Research, Vol. IV, No.3, 2006.

[8] Sunil Kumar, R K Chauhan, Parveen Kumar, "*A Minimum-process Coordinated Checkpointing Protocol for Mobile Computing Systems*", International Journal of Foundations of Computer science,Vol 19, No. 4, pp 1015-1038 (2008).

[9] Cao G. and Singhal M., "*On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing"*, in the Proc. of the Int'l Conf. on Parallel Processing, pp.37-44, Aug. 1998.

[10] Neogy S., Sinha A., and Das P., "*Distributed Checkpointing using Synchronized Clocks*," in the Proc. the 26th IEEE Annual Int'l Conf. Computer Software and Applications (OMPSAC'02), pp. 199-206, 2002.

[11] Lin Chi-Yi, Way Szu-Chi and Kuu SyoYen, "*An Efficient Time-Based Checkpointing Protocol for Mobile Computing Systems over Mobile IP*", Mobile Networks and Applications (Kluwer Acdemic Publisher Netherland), pp. 687-697, 2003.

[12] Tsai J., "*An Efficient Index-Based Checkpointing Protocol with Constant size Control Information on Messages*," IEEE Trans. on Dependable and Secure Computing, Vol. 2, No. 4, pp. 278-296, Oct-Dec 2005.

[13] Najib A. Kafahi, Said AI-Bokhitan and Ahmed AI-Nazer, "*On Disk-based and Diskless Checkpointing for Parallel and Distributed Systems*", An Empirical Analysis, Information Technology Journal, Vol. 4(4), pp. 367-376, 2005.

[14] Mandal P.S. and Mukhopadhyaya K., "*Performance Analysis of Different Checkpointing and Recovery Schemes using Stochastic Model*" Journal of Parallel and Distributed Computing, No.66, pp. 99-107, 2006.

[15] Lee, P.A., and Anderson T., "*Fault Tolerance Principles and Practice Edition*", Springer-Verlag, New York, 1990.

[16] Randell B., "*Reliable Computing Systems*", Operating Systems: An Advanced Course, Springer-Verlag, New York, pp.282-391, 1979.

[17] Gadiraju S. and Kumar Vijay, "*Recovery in the mobile wireless environments using mobile agents*", IEEE Trans. on Mobile Computing, Vol. 3, June 2004.