An Enhanced MSS-based Checkpointing Scheme for Mobile Computing Environment

M. Reza Salehnamadi* , Mohammad Hadi Asgari

Department of Computer Engineering, Postgraduate Engineering Centre Islamic Azad University, South Tehran Branch, Tehran, Iran

Received 3 December 2010; revised 11 March 2011; accepted 25 May 2011

Abstract

Mobile computing systems are made up of different components among which Mobile Support Stations (MSSs) play a key role. This paper proposes an efficient MSS-based non-blocking coordinated checkpointing scheme for mobile computing environment. In the scheme suggested nearly all aspects of checkpointing and their related overheads are forwarded to the MSSs and as a result the workload of Mobile Hosts (MHs) will reduce substantially. Moreover, the total amount of exchanging checkpoint requests will be decreased in order to have a batch transmission of such requests. The scheme is also enhanced using a simple data structure to have fewer propagating checkpoint requests and avoid the avalanche effect in the system. Simulation results show that compared to other existing algorithms, in the proposed scheme the average number of propagating requests and checkpoints, the average elapsed time for each checkpointing process, and the size of system messages are significantly lower and smaller, respectively. Thus considering its distinguishing features, the proposed approach would be efficient and suitable for mobile computing environment.

Keywords: mobile computing; fault tolerance; coordinated checkpointing; MSS-based; non-blocking.

1. Introduction

Various applications of distributed systems are developing rapidly. In addition, the idea of running the processes of the distributed systems on the mobile hosts (MHs), which will lead to developing the concept of mobile computing environment, has been attracting much more attention recently. In such an environment, the processes running on the MHs communicate with each other only by sending messages routed exclusively in the Mobile Support Stations (MSSs) which are interconnected by a fixed wired network, and therefore there is not a global shared memory or a global physical clock among the system processes. On the other hand, a cell is a geographical area covered by an MSS, and so an MH can directly communicate with an MSS through a wireless channel only if it is geographically in the cell serviced by that MSS.

Mobility requiring different hand-off and routing strategies, low bandwidth of wireless communication channels, lack of stable storage on the MHs, and disconnections and limited battery power of the MHs are some of the restrictions of mobile computing environment.

The wireless network connection in a mobile environment is more fragile compared to the fixed wired architecture of distributed systems. Thus for mobile computing systems, it must be equipped with a recovery facility. Checkpointing and rollback recovery is one of the most significant approaches for providing fault tolerance in distributed systems. But considering the restrictions of mobile computing environment, traditional checkpointing techniques which have been proposed for distributed systems are not suitable.

Non-blocking coordinated checkpointing as one of the most interesting groups of checkpointing protocols for distributed systems have been recently capturing more attention in the retrofitting mobile environment as well [3], [8], [12]. Since it can guarantee that the system will be consistent after recovery under the assumption of FIFO communication channels [4], [5], and no other process will

* Corresponding Author. Email: m_saleh@azad.ac.ir

be also blocked during the global checkpoint construction, it will be fascinating for mobile systems. In particular, if there exists a minimum number of processes involved in a coordinated checkpointing session, that would be those processes which have communicated with the checkpoint initiator either directly or indirectly since the last checkpoint [11]. Obviously, the blocking types of coordinated checkpointing are not suitable for mobile systems, considering the issues of their structures.

Communication-induced checkpointing (CIC) as another group of checkpointing protocols and their various derivatives are common in distributed systems and also in mobile environment. CIC protocols have been classified into two main types of model-based and index-based. Several algorithms based on the index-based CIC have been proposed for mobile systems [1], [3], [15]. Although there is no distinct message as checkpoint request in these protocols and therefore they will not impose the overhead of such messages to the system, there is a major incompetency in general CIC protocols. In fact, there is the possibility of creating orphan processes in a system with taking advantage of them [5].

Several checkpointing algorithms have been proposed for mobile computing systems based on checkpointing schemes. Since checkpoints must be stored in the MSSs due to the lack of stable storage in the MHs, and also because of the bandwidth limitations of such systems, an algorithm that takes fewer or no useless checkpoints in a system would be much more efficient. Moreover, decreasing the amount of propagating checkpoint requests is considered another aspect of checkpointing efficiency.

Among the existing proposed schemes, some have a more practical attitude while some others just have a theoretical or mathematical perspective. For example in [3] the authors proposed a checkpointing approach for mobile computing environment based on practical issues. Thus the proposed approach is a combination of non-blocking coordinated checkpointing and index-based CIC, in which the system initially would take mutable or tentative checkpoints for each process according to the proposed algorithm. Then if a mutable checkpoint is not converted to a tentative one, it would be redundant. The authors claimed that the scheme would lead to few number of redundant checkpoints, but after a precise consideration of the algorithm it will be clear that one of the most important causes of such few redundant checkpoint creations is the small number of mutable checkpoints which is created in the checkpointing process. Therefore, it seems that the complexity of this algorithm could be reduced without a considerable loss of its reasonable benefits. In another study [2], the author proposed an approach for dependency tracking in mobile computing environment based on some mathematical concepts including graphs. But it seems that the practical aspects of such systems are not mentioned. In other words, there is no guarantee for a system to be consistent after recovery using this approach since the generated dependency list would be altered after its creation and before being used for recovery purposes according to the conditions of real systems.

In this paper, we present an MSS-based non-blocking coordinated checkpointing scheme as there are many valid reasons behind the efficiency of MSS-based approaches with all of them coming from the fact that the MSSs play a key role in the MHs communications inside a mobile computing environment. Using this approach in a mobile system, there would be a large decrease in the number of checkpoint request messages in such a system.

Additionally, to further reduce the amount of cascading checkpoint requests, we present a simple and lightweight data structure. By manipulating and piggybacking the data structure on such requests which are sent from an MSS to another, this goal will be accomplished and also the system will be released from the avalanche effect.

The rest of the paper is organized as follows. Section 2 develops some assumptions about the features of the system model employed in this study. In Section 3, the proposed MSS-based non-blocking checkpointing algorithm for mobile computing systems is presented in detail. Then a correctness sketch about the algorithm's consistency is given in Section 4. The results of a software simulation of the proposed approach in comparison with another famous algorithm are provided in Section 5. Finally, Section 6 concludes the paper.

2. SYSTEM MODEL

We consider a mobile computing system (Figure 1) containing a set of MHs and a relatively fewer number of MSSs [1], [10]. In a mobile computing environment, a cell is a geographical location that is covered by various services of an MSS, which is providing them to support the MHs which are in that cell. There exist wireless communication links between an MH and its supporting MSS. On the other hand, the MSSs communicate with each other using fixed high-speed communication links. All of these channels which have arbitrary but finite transmission delays are reliable and they also support FIFO communication in both directions.

The considered distributed computation consists of N processes denoted by P_0, P_1, P_2, …, P_N which are running on the MHs, and each MH runs just one of the processes at a time. Like other standard distributed systems, the processes will not share a common memory or a common clock, so they will take advantage of message passing as the only way of communicating with each other. An MH can move to another neighboring cell which is called a hand-off.

There are two kinds of messages in the system, namely computational and system messages. The computational messages are generated by the underlying distributed application and the system messages are generated by the system processes in order to generate new consistent global checkpoints.

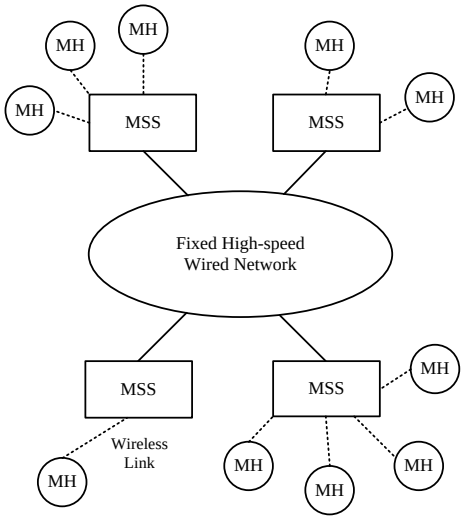

Fig. 1. Mobile Computing System Architecture

Since the MHs are typically powered by batteries, they can power down its particular components during their low activity phases for energy saving reasons [7]. This method is indicated as the sleep mode or the doze mode operation.

THE NON-BLOCKING MSS-BASED HECKPOINTING SCHEME

In this section we firstly present our basic non-blocking MSS-based checkpointing scheme and then an improvement approach to further reduce the number of messages containing checkpoint requests, which is based on generating and updating a simple data structure and piggybacking it on such requests, is described.

A. The MSS-based Approach

The MSSs play a key role in MHs communications inside a mobile computing environment. In fact, each message that has been sent by an MH must be routed among the MSSs to reach the destination MH. Furthermore, the checkpoints must pass from the MHs to the MSSs in order to be stored in their stable storage because the MHs do not have such a reliable storage as mentioned before. So it seems that the approach suggested here make the transfer of almost the total overhead of checkpointing and rollback recovery and their related algorithms and data structures from the MHs to the MSSs possible. It is also much more efficient than the other approaches that do not have such an operation inside themselves. Moreover, as the MSSs are more appropriate for data storage than the MHs according to their hardware and software structures, it is better to store the data structures in the MSSs rather than in the MHs. After all, in this approach we could have the batch sending and receiving of checkpoint requests from an MSS to another, and this would greatly decrease the amount of such messages in the system which is highly desirable in mobile environment as well as other types of distributed systems.

The general description of our approach is as follows. While each MH is running its own computations and passing messages to other MHs in order to advance computations of the general application, one of the MHs that finds itself ready to initiate a checkpointing process informs its supporting MSS about the situation and sends its current state to it. When the MSS receives the initiator MH's request, it stores the initiator MH's state as its temporary checkpoint and then forwards it to other MSSs that are supporting the MHs related to the initiator MH. In other words, the MSS generates packages containing checkpoint requests for each MSS that supports at least one MH related to the initiator, and then it sends them to the corresponding MSSs. This batch style of request sending, which would be utilized in other aspects of the proposed scheme too, greatly decreases propagation of such checkpoint requests. At the other point of request transition, the receiver MSS sends the same requests to other MSSs which are supporting the MHs related to each of its desired MHs according to the received package's information in a batch style too after it takes a temporary checkpoint for each one of its desired MHs. Clearly, since an MSS must receive an MH's status from itself in order to store it as that MH's checkpoint in its permanent storage, it may send a request to that MH and receive its status in response to the request whenever it needs. This temporary manner of taking checkpoints and batch request propagation process continues until all the MHs related to the initiator are covered. Thus when the initiator MH's supporting MSS is informed about this situation, it broadcasts a commit message, again in a batch style, to other MSSs, and then each MSS makes its respective temporary checkpoints permanent by receiving the commit message, and removes its former checkpoints which are not useful anymore. As a result, a global consistent system state is provided using this approach.

B. The Improved MSS-based Checkpointing Scheme

The basic proposed scheme may lead to the avalanche effect in which there would be a recursive request sending condition in a system. In order to have an avalanche effect free scheme, we introduce a simple data structure containing the identification numbers of the MHs which have been mentioned in a checkpoint request sending operation by another MSS before. Each MSS manipulates its copy of such data structure to complete it according to its future request sending operations, and piggybacks it on such messages. When the receiver MSS catches a copy of such a data structure, it notices that it should not attempt to send a request relating to its aforementioned MHs, and thus complete it using its future request sending operations, and then piggyback it on such messages. Other aspects of the scheme are exactly like the basic one above.

C. Notations and Data Structures

In this section we will present a basic description about the main concepts and data structures of our proposed algorithm:

1) MH_Data: The data structure containing the MHs' required data, which is stored in the MSSs. This complex structure consists of the following variables:

- *Is_In_This_Cell*: A flag which is set if the corresponding MH is in the MSS's cell. Its default value is false.

- *R* (Received): An array almost like the one in [3] but here each block of it is related to an MH. So this array has N blocks, in which N is the total number of the system's MHs. According to the block index number, the value of each block of this array which is related to its corresponding MH is 1 if the corresponding MH has sent at least one computational message to this MH in the current interval and 0 if it has not. All of the array's blocks would be set to 0 as their initial value.
- *weight*: A nonnegative variable of real type with a maximum value of 1. We use it to detect the termination of the checkpointing algorithm as in [9]. Its default value is 0.
- *Initiator_ID*: This variable indicates the identification number of the checkpointing initiator MH that started the latest checkpointing process. We assume that at any time there will be a single checkpoint initiation. Since the identification numbers of MHs starts from 1, and 0 does not belong to any MH, the initial value of Initiator_ID would be 0.

2) not_sent_counter: This counter is utilized to find the special case in which the initiator MH does not receive any messages from any other MHs in the interval. In this situation there is no need for any other MSSs to store the state of some of their supporting MHs as their checkpoints, except for the initiator's MSS.

3) Sent_Req: A data structure that is put to practical use in order to decrease the amount of propagating checkpoint requests and also to have an avalanche effect free approach. The MSS which is supporting the initiator MH generates a new copy of the data structure and fills it with the relevant information according to its checkpoint request activities and piggybacks it for the MSSs that it should. So such MSSs will receive a copy of this data structure and use and update it and then piggyback it for the other relevant MSSs. Sent_Req consists of these variables:

- *CP_Initiator*: An integer variable containing the identification number if the MH triggered current checkpointing process.
- *Req_Deliv_List*: A binary array of size N, in which N is the total number of the system's MHs. If the MSS notices that it must send a checkpoint request related to an MH to its supporting MSS, it will set the flag having an index number equal to that MH's identification number.

4) CSMHW (Corr_Supp_MHs_and_weights): An array that has M blocks, in which M is the total number of the system's MSSs. Each block of this array points to a list (e.g. an arraylist) containing the information about some of MSSs supporting the MHs which are related to at least one of the sender MSS's supported MHs, and their weights. This data structure is utilized to have batch checkpoint request transmission in the system. Each element of the list, which each block of this array points to, consists of these variables:

- *MH_ID*: An integer variable containing the identification number of the corresponding MH.
- *MH_Weight*: A nonnegative variable of real type, containing a portion of the sender MSS's respective supported MH's weight that it allocated to the corresponding MH.

5) SPhCSMHW (Sec_Ph_Corr_Supp_MHs_and_weights): This data structure is utilized to create batch second phase responding in the system and its type and variables are exactly the same as CSMHW's.

D. The Enhanced MSS-based Algorithm

In this section we present a pseudo code of the proposed algorithm to provide a more clear understanding of it. The pseudo code will be presented from two different perspectives of MHs and MSSs.

1) MHs Participation in the algorithm

As mentioned before, nearly all the overhead of this scheme is forwarded to the MSSs, thus participation of the MHs in this checkpointing algorithm is very low. In other words, they would just send requests to their supporting MSSs in order to send computation messages or checkpoint initiation, and in the case of receiving a computation message they just need to process it and there is no need for any kind of information storage.

MH_m on sending a computation message to MSS_p (its supporting MSS) in order to deliver it to MSS_q (MH_n's supporting MSS):

```
Send_Comp_Message(MH_n)
{
   Send the computation message, containing (MH_m, MH_n) to MSS_q;
}
```

MH_m on receiving a computation message from MSS_p (its supporting MSS) which has been sent by MSS_q (MH_n's supporting MSS):

```
Receive_Comp_Message(MH_n)
{
   Process the message (no other activity needed);
}
```

MH_m on finding itself ready to initiate a checkpointing process through MSS_p (its supporting MSS):

```
Initiate_Ckp()
{
   Send checkpoint initiation request, containing (MH_m) to MSS_p;
   Send its local state as the checkpoint to MSS_p;
}
```

2) MSSs Participation in the algorithm

As the MSSs are responsible for providing correct checkpointing processes in the system, they are in charge of storing needed information about the MHs in the related data structures and also other relevant activities. Since it could be possible for the MHs to leave a cell and enter a new cell (hand-off), the proposed algorithm also include some pertinent functions to meet hand-off requirements properly.

MSS_p on receiving a checkpoint initiation request from MH_m (one of its supported MHs):

```
Initiate_Checkpoint(MH_m)
{
   MH_Data[m].Initiator_ID = m;
   MH_Data[m].weight = 0;
   Sent_Req.CP_Initiator = m;
   Store MH_m's state as its temporary checkpoint;
   Spread_CP_Req(MH_Data[m].R, MH_m, Sent_Req,
   MH_Data[m].Initiator_ID, 1.0);
   Reset MH_Data[m].R;
```

```
}
Spread_CP_Req(R_m, MH_m, Sent_Req, MH_Init_ID, received_weight)
{
  MH_Data[m].weight = received_weight;
  for(i=1 to the number of MHs)
  if((R_m[i] == 1) && (Sent_Req.CP_Initiator == MH_Init_ID)
    && (Sent_Req.Req_deliv_List[i] == 0))
  {
    MH_Data[m].weight = MH_Data[m].weight / 2;
    CSMHW[MH_i's current supporting MSS's ID]
      .add(MH_i, MH_Data[m].weight);
    Sent_Req.Req_deliv_List[i] = 1;
  }
  else
    not_sent_counter++;
  for(j=1 to the number of MSSs)
  if(CSMHW[j] is not empty yet)
    Send Checkpoint Request to MSS_j, containing(Sent_Req,
      CSMHW[j], MH_Data[m].Initiator_ID);
  if((not_sent_counter == total number of MHs) && (m == MH_Init_ID))
  Broadcast(commit, MH_Init_ID) to all MSSs;
  not_sent_counter = 0;
}
```

MSS$_p$ on receiving a checkpoint request from MSS$_q$ (another MSS) on behalf of MH$_m$ (one of MSS$_q$'s supported MHs):

```
Receive_Checkpoint_Request(Sent_Req, Rec_CSMHW, MH_Init_ID)
{
  for(i = 0 to the size of Rec_CSMHW)
  if (MH_Init_ID == MH_Data[Rec_CSMHW.get(i).MH_ID].Initiator_ID)
    SPhCSMHW.add(Rec_CSMHW.get(i).MH_ID,
      Rec_CSMHW.get(i).MH_Weight);
  else
  {
    MH_Data[Rec_CSMHW.get(i).MH_ID].Initiator_ID = MH_Init_ID;

  Spread_CP_Req(MH_Data[Rec_CSMHW.get(i).MH_ID].R
  ,
      Rec_CSMHW.get(i).MH_ID,
      Sent_Req, MH_Init_ID,
  Rec_CSMHW.get(i).MH_Weight);
    Store the state of the MH with the ID
  Rec_CSMHW.get(i).MH_ID
      as its temporary checkpoint, after requesting and
  receiving its state;
    SPhCSMHW.add(Rec_CSMHW.get(i).MH_ID,
      MH_Data[Rec_CSMHW.get(i).MH_ID].weight);
    Reset R of the MH with the ID
  Rec_CSMHW.get(i).MH_ID;
  }
  Send Second Phase Response to the supporting MSS of the
  initiator MH
  with the ID MH_Init_ID, containing(MH_Init_ID,
  SPhCSMHW);
}
```

MSS$_p$ on receiving a computation message sending request from MH$_m$ (one of its supported MHs) to MH$_n$:

```
Send_Comp_Message(MH_m, MH_n)
{
  Send the computation message, containing(MH_m, MH_n)
  to MH_n's supporting MSS;
}
```

MSS$_p$ on receiving a computation message sending to MH$_m$ (one of its supported MHs) request from MSS$_q$ on behalf of MH$_n$ (one of its supported MHs):

```
Receive_Comp_Message(MH_n, MH_m)
{
  MH_Data[m].R[n] = 1;
  Send the computation message, containing(MH_n) to MH_m;
}
```

MSS$_p$ on receiving a second phase response from MSS$_q$, related to MH$_m$ (the checkpoint initiator):

```
Receive_Sec_Ph_Resp(MH_m, Rec_SPhCSMHW)
{
  for(i = 0 to the size of Rec_SPhCSMHW)
  {
  MH_Data[m].weight = MH_Data[m].weight +
    Rec_SPhCSMHW.get(i).MH_Weight;
  if(MH_Data[m].weight == 1.0) Broadcast(commit, MH_m) to
  all MSSs;
  }
}
```

MSS$_p$ on receiving a broadcasted commit message, which is related to MH$_n$ (the checkpoint initiator)

```
Receive_Broadc_Msg(commit, MH_n)
{
  for(All supported MHs)
  {
  Make all temporary checkpoints of supported MHs
  permanent;
  Delete all checkpoints of supported MHs, except each one's
  last checkpoint;
  }
}
```

MSS$_p$ on facing the departure of MH$_m$ from its area to MSS$_q$'s (another cell)

```
Send_MH_Data(MH_m, MSS_q)
{
  Send MH_m's data, containing(MH_m, MH_Data[m]) to
  MSS_q;
  MH_Data[m].Initiator_ID = m; Reset MH_Data[m].R;
  MH_Data[m].weight = 0; MH_Data[m].Is_In_This_Cell =
  false;
}
```

MSS$_p$ on receiving the data of MH$_n$ in order to face its entry from MSS$_q$'s area

```
Receive_MH_Data(MH_n, MH_n_Data)
{
  MH_Data[n].Initiator_ID = MH_n_Data.Initiator_ID;
  MH_Data[n].R = MH_n_Data.R;
  MH_Data[n].Is_In_This_Cell = true;
  MH_Data[n].weight = MH_n_Data.weight;
}
```

CORRECTNESS SKETCH

As mentioned before, the proposed algorithm is based on the coordinated checkpointing scheme which is completely free of orphan messages and orphan processes and therefore it always makes consistent system states [5]. Even non-blocking coordinated schemes would be free of inconsistent checkpoints under the assumption of FIFO communication channels [4], [5]. There are also other

alternative solutions to have consistent non-blocking coordinated schemes without the assumption of FIFO channels. Piggybacking markers which are some kind of checkpoint requests on every post-checkpoint message [13], or checkpoint indices which can serve the same role as markers [6], [14] and are partly similar to the actions utilized in the CIC schemes are some of these alternative methods.

Although the concept of having batch checkpoint requests seems to be beyond the scope of standard coordinated checkpointing schemes, we should argue about the scheme's consistency with regard to this idea. If we assume a set of messages which is being sent by an MSS to another MSS as a single but large message, and then reanalyze the algorithm with such an idea, we would find out that the algorithm exactly matches with the standard coordinated checkpointing methods which we argued about their consistency before. Now as we assumed that we have FIFO channels in our system and also as the transposition of different request messages in a set of system messages which is being sent to another MSS obviously is not related to the system's consistency, by extending the large message assumption to a set of messages, we can find out that the algorithm does not disrupt the system's consistency after the checkpointing process.

A Performance Evaluation Based on Simulation

In order to evaluate our proposed algorithm and obtain a worthwhile comparison between the algorithm and other existing algorithms, we take advantage of the software modeling and simulation. In other words, we simulate a mobile computing system with a number of MSSs and a relatively larger number of MHs which are randomly placed in these MSSs initially, and each one of them can leave a cell and enter another cell over time (hand-off) with the time intervals following an exponential distribution. The MHs send messages to each other through their supporting MSSs again with the time intervals following an exponential distribution. 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 reachs 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 also implement the Cao and Singhal's algorithm [3] as one of the most famous and fairly efficient algorithms in the field of mobile computing systems' fault tolerance based on checkpointing and rollback recovery according to their published information. Thus we can compare the results of our proposed algorithm with those of the Cao and Singhal's algorithm.

The average number of the propagating checkpoint requests and checkpoints, the average elapsed time for each checkpointing process (in milliseconds) and the average size of system messages (in bytes) are four important factors of checkpointing algorithms in distributed systems and in mobile computing environment as well. Figures 2 to 5 show the simulation results in relation to these four factors in different message sending rates.

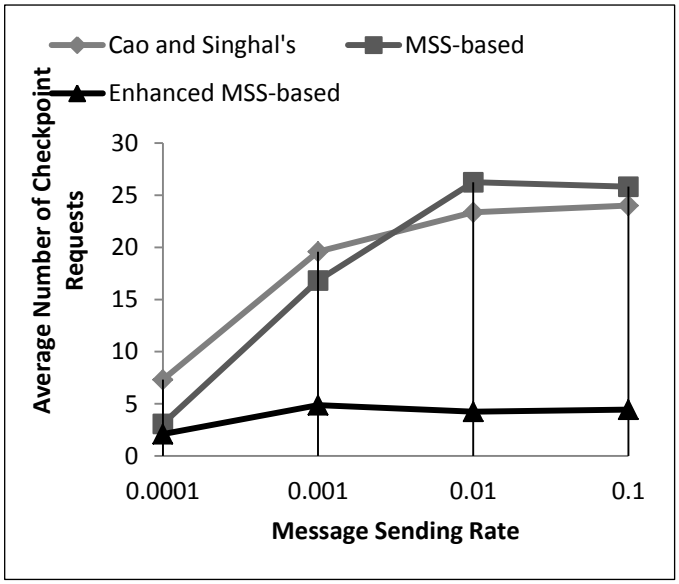

Fig. 2. The number of checkpoint requests

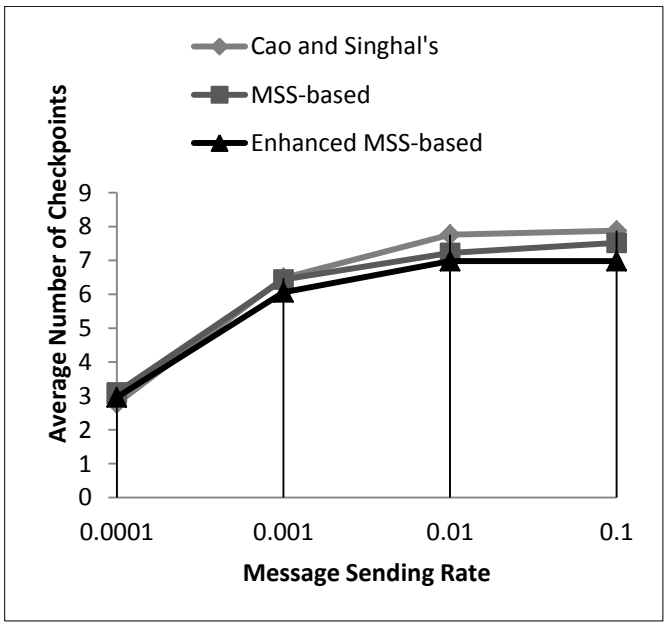

Fig. 3. The number of checkpoints

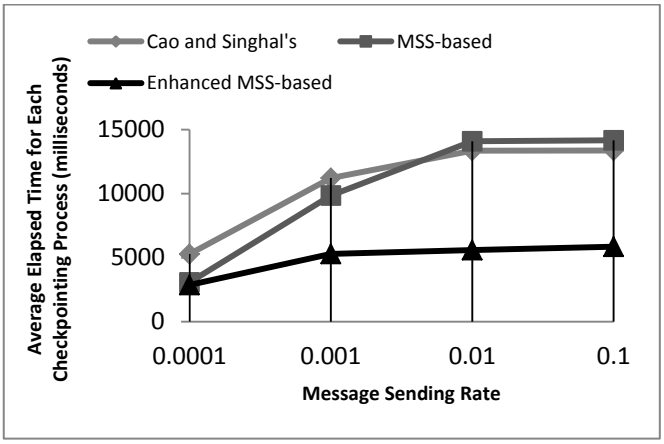

Fig. 4. The elapsed time for each checkpointing process (milliseconds)

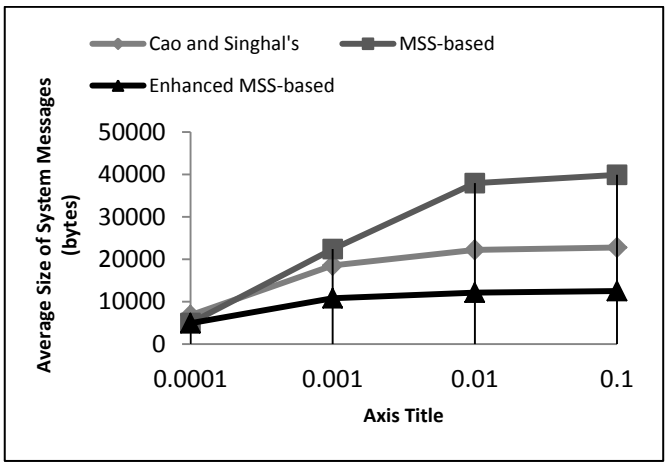

Fig. 5. The size of system messages (bytes)

CONCLUSIONS

In this work, we presented an MSS-based checkpointing scheme for mobile computing systems, and then an enhancement for the scheme using a simple and lightweight data structure. In the approach suggested here, the imposed overhead of the checkpointing and its entire ingredients (data structures, protocols, etc.) would be transferred to the MSSs so that the MHs will have no concerns about them. Batch checkpoint request transmission which is possible is utilized in this scheme too, thus the amount of such messages would decrease to a considerable extent in the system. On the other hand, since our scheme is based on non-blocking coordinated checkpointing approaches, the system would take a minimum number of checkpoints in comparison with the other algorithms which are not based on such protocols or take advantage of other approaches in addition. Moreover, in the proposed algorithm there is no possibility of orphan message creation since it fully complies with the coordinated checkpointing approaches which are orphan-free. The advantages mentioned indicate that the proposed checkpointing scheme presents a more practical perspective of tolerating mobile computing systems against faults, and is also efficient and suitable for such systems.

REFERENCES

[1] A. Acharya and B. R. Badrinath, Checkpointing distributed applications on mobile computers, Proceedings of the Third International Conference on Parallel and Distributed Information Systems, 1994.

[2] S. Bhalla, Independent dependency tracking in a mobile adhoc computing environment, First International Conference on Communication System Software and Middleware (Comsware), 2006.

[3] G. Cao and M. Singhal, Mutable checkpoints: a new checkpointing approach for mobile computing systems, IEEE Transactions on Parallel and Distributed Systems, Vol. 12, No. 2, February 2001.

[4] M. Chandy and L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Transactions on Computing Systems, Vol. 31, No. 1,pp. 63-75, 1985.

[5] E.N. Elnozahy, L. Alvisi, Y.M. Wang, and D. B. Johnson, A survey of rollback-recovery protocols in message-passing systems, ACM Computing Surveys (CSUR), Vol. 34 , No. 3, pp. 375-408, 2002.

[6] E.N. Elnozahy, D.B. Johnson, and W. Zwaenepoel, The performance of consistent checkpointing, Proceedings of Eleventh Symposium on Reliable Distributed Systems, pp. 39-47, 1992.

[7] G.H. Forman and J. Zahorjan, The challenges of mobile computing, IEEE Computer, pp. 38-47, April 1994.

[8] B. Gupta, S. Rahimi, R. A. Rias, and G. Bangalore, A low-overhead non-block checkpointing algorithm for mobile computing environment, GPC 2006, LNCS 3947, pp. 597-608, 2006.

[9] S.T. Huang, Detecting termination of distributed computations by external agents, Proc. Ninth Int'l Conf. Distributed Computing Systems, pp. 79-84, 1989.

[10] T. Imielinski and B.R. Badrinath, Mobile wireless computing, Commun. ACM, Vol. 37, No. 10, pp. 18-28, Oct. 1994.

[11] R. Koo and S. Toueg, Checkpointing and rollback-recovery for distributed systems, IEEE Transactions on Software Engineering, Vol. 13, No. 1, pp. 23-31, 1987.

[12] L. Kumar, M. Mishra and R.C. Joshi, Low overhead optimal checkpointing for mobile distributed systems, Proceedings of the 19th International Conference on Data Engineering (ICDE'03), 2003.

[13] T.H. Lai and T.H. Yang, On distributed snapshots, Information Processing Letters, Vol. 25, pp. 153-158, 1987.

[14] L.M. Silva, Checkpointing mechanisms for scientific parallel applications, Ph.D. Thesis, University of Coimbra, Department of Computer Science, 1997.

[15] J. Yang, J. Cao and W. Wu, Efficient global checkpointing algorithms for mobile agents, Concurrency and Computation: Practice and Experience, Vol. 20, pp. 825-838, 2008.