International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:8, 2007

On Mobile Checkpointing using Index and Time Together

Awadhesh Kumar Singh

***Abstract*—**Checkpointing is one of the commonly used techniques to provide fault-tolerance in distributed systems so that the system can operate even if one or more components have failed. However, mobile computing systems are constrained by low bandwidth, mobility, lack of stable storage, frequent disconnections and limited battery life. Hence, checkpointing protocols having lesser number of synchronization messages and fewer checkpoints are preferred in mobile environment. There are two different approaches, although not orthogonal, to checkpoint mobile computing systems namely, time-based and index-based. Our protocol is a fusion of these two approaches, though not first of its kind. In the present exposition, an index-based checkpointing protocol has been developed, which uses time to indirectly coordinate the creation of consistent global checkpoints for mobile computing systems. The proposed algorithm is non-blocking, adaptive, and does not use any control message. Compared to other contemporary checkpointing algorithms, it is computationally more efficient because it takes lesser number of checkpoints and does not need to compute dependency relationships. A brief account of important and relevant works in both the fields, time-based and index-based, has also been included in the presentation.

***Keywords*—**Checkpointing, forced checkpoint, mobile computing, recovery, time-coordinated.

I. Introduction

ROLLBACK recovery is essential in fault-tolerant distributed computing. Checkpointing is used as a popular technique for rollback recovery since Chandi-Lamport introduced distributed snapshots [1]. A checkpoint is a snapshot of the local state of a process, saved on local non-volatile storage to survive process failures [2]. The purpose of checkpointing is to reduce recovery time by providing an earlier database state. The mobile hosts, roaming between places, make mobile computing systems more prone to frequent process failures due to physical damage, like lost and stolen, and transient failures like power and connectivity problems [3]. A local checkpoint includes the local state of one process, whereas, a global checkpoint includes the local states of all the processes in the system, provided one from each process. A message exchanged between a pair of processes is called orphan if its receive event has already been recorded in the local checkpoint of receiver process while its send event is yet to be recorded in the local checkpoint of sender process. The existence of orphan messages pose consistency problem in the system. A global checkpoint is consistent if and only if no pair of its constituent checkpoints has any orphan message. The well-known problem of unbounded *domino effect* [4] arises if processes take their local checkpints independently. Hence, the checkpointing protocols using some coordination mechanism are more popular.

Two methods of coordination have been proposed in the literature [5] (a) Coordinated checkpointing, where protocols have to exchange explicit control messages during checkpoint creation to save consistent global states. Although, this should be avoided because of frequent disconnections in the mobile computing environment. (b) Communication-induced checkpointing (CIC), where protocols piggyback control information on application messages, thereby, avoids addition of explicit control messages, to the computation, during checkpoint creation [6]. In addition to basic checkpoints, that is, local checkpoints taken by each process independently at its own pace, the protocol-specific information piggybacked on application messages may "induce" each process to take some extra checkpoints, termed forced checkpoints, before it can process the messages. Basically, forced checkpoint is "induced" by a condition, which is tested after receiving every communication (message). The class of protocols derives its name "communication-induced" from this fact only.

CIC protocols allow processes to enjoy greater autonomy, compared to their other checkpointing counterparts, in deciding when to take local checkpoints. However, the increased number of forced checkpoints negates the "autonomy" advantage substantially. Therefore, the research efforts are mostly directed towards limiting the number of forced checkpoints to the fullest possible extent.

There are two different types of Communication-induced protocols. (a) model-based, where checkpointing protocol tries to avoid the domino effect and mimic a piece-wise deterministic behavior for each process [7]. (b) index-based, where a sequence number is assigned to each local checkpoint. Associating every local checkpoint with a sequence number is, basically, timestamping every local checkpoint with Lamport's logical clock [8]. Any global checkpoint, consisting of local checkpoints with the same sequence number, is guaranteed to be consistent. Index-based protocols are preferred over their model-based counterparts because they piggyback smaller control information on application messages and force lesser checkpoints [9], [6].

Author is with the Department of Computer Engineering, National Institute of Technology, Kurukshetra, 136119, India (e-mail: aksinreck@rediffmail.com).

The present work is also about the design and analysis of an index-based checkpointing protocol, which uses time coordination in order to reduce the number of total checkpoints. The paper is organized as follows: Section 2 describes system model, section 3 gives algorithm concept, and section 4 presents the algorithm. Section 5 contains a brief account of important related work. Finally, section 6 concludes the work.

II. System Model

A mobile computing application is executed by a set of n processes, denoted by P_1, P_2,…P_n, running on several mobile hosts (MHs). Processes communicate with each other by sending messages. Each process progresses at its own speed and messages are exchanged through an asynchronous and reliable channel with unpredictable but finite transmission delays, that is, no message will be lost in the channel. The messages, originated from a source MH, are received by the local mobile support station (MSS) and then forwarded to the destination MH. All MSSs are interconnected by a fixed network. The MSSs may belong to different organizations. A typical wireless cell is shown in following Fig. 1.

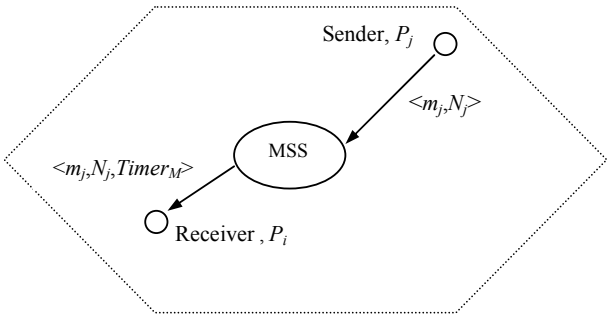

Fig. 1 The wireless cell

All processes in the system take their checkpoints periodically. Before a mobile computing application starts, a predefined checkpoint period T is set on the timers. When the local timer expires, the process saves its system state as a checkpoint and assigns it a checkpoint sequence number. Any subsequent checkpoint is assigned higher checkpoint sequence number, which increases monotonically. The time interval between the k^{th} checkpoint and k+1th checkpoint is called the kth checkpoint interval.

Every MH and MSS contains a system clock, with typical clock drift rate ρ in the order of 10^{-5} or 10^{-6}. The system clocks of MSSs can be synchronized using internet synchronization services such as network time protocol (NTP), which makes the maximum deviation of all the clocks within tens of milliseconds. Timers in MSSs are used as reference because MSSs, being fixed hosts, have more reliable timers than those in MHs. Every application message m_j is piggybacked with two values viz. checkpoint sequence number N_j of the sender P_j and local timer of the MSS $Timer_M$, that is, time to next checkpoint. The MSS, closest to the receiver, is responsible for piggybacking its local timer $Timer_M$, time to next checkpoint, in every application message m_j destined to the receiver.

III. Algorithm Concept

We assume that each process takes an initial basic checkpoint and that, for the sake of simplicity, basic checkpoint is taken by a periodic algorithm [7]. Every process sends the snapshot of its initial state (termed as 0th checkpoint) to its local MSS when the application starts. A checkpoint period T, predefined by local MSS, is set on the timers of all the mobile hosts before starting the mobile computing application. For any process, whenever its local timer expires, it takes a checkpoint if it has sent any message in the current checkpoint interval. If no message has been sent in the current checkpoint interval then it will not take the checkpoint for the current checkpoint interval.

However, due to varying drift rates of local clocks, the timers at different sites are not perfectly synchronized. Hence, the checkpoints may not be consistent because of orphan message. In order to avoid this situation, every message sent between processes is piggybacked with the sender's information which tells how many checkpoint intervals have passed at the sender process. Using this information, creation of orphan messages is avoided.

Every message, originated from a process, reaches its destination through the MSS, where it is piggybacked 'time to next checkpoint' by the local MSS. When the message is received by the receiver, it sets its local timer equal to the timer of local MSS. In this way, the timer synchronization is implemented.

After taking checkpoint, processes send their checkpoint information to its local MSS, where it is stored in a stable storage. A global checkpoint consists of all the Nth checkpoints of every process, where $N \geq 0$. If any process has not taken its Nth checkpoint (as it did not send any message in the Nth checkpoint interval), its previous checkpoint would be included in the Nth global checkpoint. The Nth global checkpoint is not complete unless every process sends either Nth checkpoint or the information that N checkpoint intervals have passed.

A. Choosing Checkpoint Period

In time-based checkpointing protocols, the length of recovery is directly proportional to the checkpoint period T. Larger checkpoint periods result, on average, in larger rollbacks, and consequently in longer recovery times [10]. Hence, shorter checkpoint periods are preferable. However, the checkpointing protocol can only begin to create a new checkpoint when the previous checkpoint has been completely stored on the stable storage. Thus, the checkpoint period should be greater than the time taken to save the checkpoint completely on the stable storage. Therefore, the checkpoint period is lower bounded by the checkpoint latency. The checkpoint latency can be defined as the time a protocol takes

to save a new checkpoint [11], [12].

B. Working of the Algorithm

The algorithm uses the following local variables:

c_i : a boolean variable that is initialized as 0. If a message is sent by a process P_i in the current checkpoint interval, c_i is set to 1.

N_i : number of checkpoint intervals passed.

$Timer_i$: timer of process P_i.

$Timer_M$: timer of the MSS.

flag: if it is set to 0 process takes a checkpoint after expiry of its local timer. No, otherwise.

Process P_i takes a checkpoint when either it's local timer $Timer_i$, has expired or when it receives a piggybacked message m_j from some other process P_j. If $Timer_i$ has expired then it checks whether c_i =1. If it is true then the process takes a checkpoint otherwise continues its normal operation. If a process P_i receives a piggybacked message m_j but its local timer $Timer_i$ has yet not expired then it checks whether N_j of the sender P_j is greater than N_i of the receiver P_i. There are two possible cases: (1) If it is true P_i sets the *flag* variable (*flag* =1) and takes a checkpoint if c_i =1, without recording the current reception event. Afterwards, P_i processes the received message m_j. As *flag* is set to 1, P_i doesn't take checkpoint further, in the current checkpoint interval, after expiry of its local timer $Timer_i$. The following Fig. 2 illustrates this scenario. (C_x^y: yth checkpoint of process P_x)

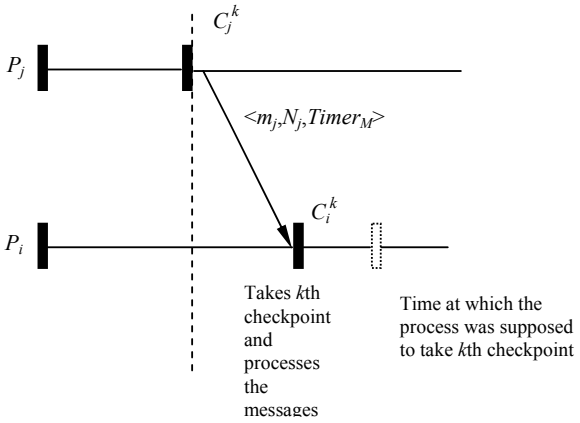

Fig. 2 Case 1 ($Nj > Ni$)

(2) If it is false then process P_i directly processes the received message m_j without taking any checkpoint. However, process P_i takes a checkpoint, in the current checkpoint interval, after expiry of its local timer if c_i =1. The following Fig. 3 illustrates this case.

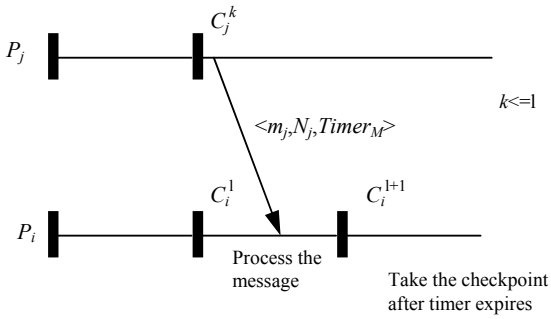

Fig. 3 Case 2 ($N_j <= N_i$)

In order to synchronize the local timers of processes, whenever the MSS forwards an application message m_j to some process P_i in its local cell, MSS piggybacks its own 'time to next checkpoint' $Timer_M$ in application message m_j and receiving process P_i synchronizes its timer accordingly.

IV. The Algorithm

Following is the pseudo code of the algorithm:

A. Pseudo Code

```
At each process Pi (1≤i≤n)
  if (Timeri expires)
    if (flag=1) then
      flag=0
      do not take checkpoint and resume
      normal computation
    else
      Ni=Ni+1
      if (ci=1)
        take checkpoint
        ci=0
        resume normal computation
      else
        do not take checkpoint and
        resume normal computation
      endif
    endif
  else
    if (Pi receives message <mj, Nj, TimerM>)
      Timeri=TimerM
      if (Nj>Ni)
        flag=1
        Ni=Ni+1
        if (ci=1)
          take checkpoint
          ci=0
        endif
      endif
      process message mj
      resume normal computation
    else
      resume normal computation
    endif
  endif
```

B. Proof of Correctness

Theorem 1. Let P_j be any process whose $c_j = 1$ and latest checkpoint is C_j^k then none of the messages sent by P_j in the kth checkpoint interval can become orphan.

Proof. The variable c_j is set to 1 if and only if process P_j has sent at least one message (say m_j) in the current checkpoint interval and it has not taken the checkpoint for this interval yet. This implies that P_j has not recorded this event of sending message m_j.

If any receiving process P_i first processes messages m_j and then takes its kth checkpoint then message m_j may become orphan since the event of receiving message m_j is recorded in the kth checkpoint of process P_i but event of the sending of message m_j has not been recorded in kth checkpoint interval of process P_j. This would make the kth global checkpoint inconsistent. However, this never happens as process P_i, on receiving message m_j, would take its kth checkpoint first and then process the message m_j. This is implemented by piggybacking every message m_j with variable N_j of the sender P_j (N_j denotes the checkpoint intervals passed at process P_j). On the basis of value N_j, receiver P_i decides whether to process the received message m_j directly, or to take checkpoint first and subsequently process the message m_j. When process P_i receives message m_j it compares the piggybacked variable N_j with its own variable N_i. If $N_j <= N_i$, then it implies that process P_i has already taken its kth checkpoint. So, it processes the received message m_j directly. If $N_j > N_i$, then it implies that P_i has yet not taken its kth checkpoint. So, it first takes the checkpoint C_i^k (say at time t_i) and then process message m_j. Thus, the kth checkpoints, of both processes sender P_j and receiver P_i, do not record anything regarding message m_j, so kth checkpoint is consistent and message m_j cannot become orphan in the kth global checkpoint.

Process P_j will definitely take its (k+1)th checkpoint (since it has sent message m_j in the kth interval) and hence (k+1)th global checkpoint is consistent.

Theorem 2. If at any given time t, $c_i = 0$ for process P_i and C_i^k is its latest checkpoint, then none of the messages sent by P_i remains orphan at time t.

Proof. There can be two cases for process P_i under the given conditions:

Process P_i took checkpoint C_i^k exactly at time t and has reset $c_i = 0$. This implies any message sent by process P_i before time t must have been recorded up to checkpoint C_i^k. As send event of the message(s) has been recorded, none of the message(s) sent by P_i would be orphan at time t.

Process P_i took its latest checkpoint C_i^k at some instant t_j where $t_j<t$. Now, $c_i = 0$ implies that it has not sent any message since its last checkpoint at time t_j and all messages sent before t_j have already been recorded by the previous checkpoints. Hence, none of the messages sent by process P_i would be orphan at time t.

From theorem 1 and 2, we conclude that at any given time t, none of the messages sent by any process P_i would be orphan. Therefore, at any given time t, the proposed algorithm ensures the existence of globally consistent checkpoint.

Theorem 3. The algorithm terminates within a finite time.

Proof. The proof is trivial, as the algorithm does not use any shared variable and does not contain any looping structure.

C. Adaptivity

An algorithm is called adaptive when it adjusts and functions well within a changing environment. In our protocol, a process does not take a local checkpoint, scheduled shortly after a forced checkpoint, because such local checkpoint advances the recovery point of the process by a very short amount compared to the forced checkpoint just taken, however, it may generate a flurry of forced checkpoints throughout the system [9]. In order to address this problem, we increment the checkpoint sequence number on taking the forced checkpoint and do not take the local checkpoint scheduled shortly after it. In this way, by taking the forced checkpoint into account, we make our local checkpointing policy adaptive. This approach reduces the number of local checkpoints that, in turn, reduce the number of forced checkpoints.

D. Problem Associated with Time-based Approach

In [13], authors pointed out that the time-based checkpointing schemes may be inconsistent when failure occurs in the situation explained below in Fig.4. P_i and P_j are two processes executing simultaneously in a mobile computing system. Due to clock drift, process P_i takes its C_i^k checkpoint before process P_j takes its C_j^k checkpoint. Now, after these checkpoints have been taken, P_j sends a message m_j to P_i. Since N_i=N_j, P_i directly processes message m_j without taking any checkpoint. Now, suppose P_i takes checkpoint C_i^{k+1} at time t_1 and P_j takes checkpoint C_j^{k+1} at time t_2 where $t_1<t_2$. If the system crashes between time t_1 and t_2, then P_i restarts from C_i^{k+1} and P_j from C_j^k (as checkpoint C_j^{k+1} has not been taken yet). This results in inconsistency as m_j becomes orphan in this case because its send event was not recorded by P_j but the receive event has been recorded by P_i.

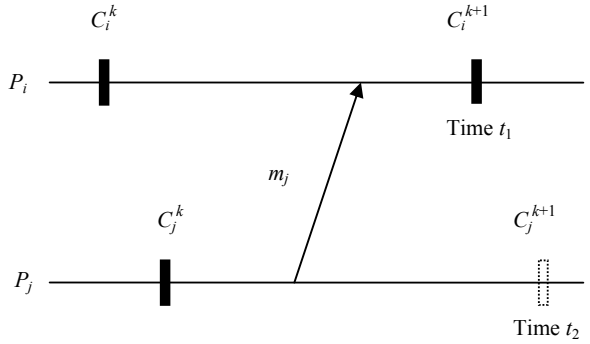

Fig. 4 A possible inconsistency

Our algorithm is also time-based; however, this inconsistency situation never arises there. A global checkpoint consists of all the Nth checkpoints of every process, where $N\geq 0$. If any process has not taken its Nth checkpoint (as it did

not send any message in the Nth checkpoint interval), its previous checkpoint would be included in the Nth global checkpoint. The Nth global checkpoint is not complete unless each process has either sent its Nth checkpoint or the information that N checkpoint intervals have passed (in this case its previous checkpoint would be included in the Nth global checkpoint). When system crashes between time t_1 and t_2, the (k+1)th global checkpoint is not complete yet as P_j has not taken its (k+1)th checkpoint. So, P_i does not restart from C_i^{k+1} but from C_i^k as kth global checkpoint is complete. In kth global checkpoint, neither the event of sending message m_j nor the event of receiving message m_j was recorded, so message m_j does not become orphan and kth global checkpoint is consistent. Hence, the common inconsistency problem for time-based checkpoint scheme, pointed out in [13], does not arise in our algorithm.

V. Related Works

We have used the concept of time-coordination in order to design an index-based checkpointing protocol. Thus, we review, one by one, both time-coordinated and index-based checkpointing protocol and compare our work with them.

A. Time-based Checkpointing Protocols

Time-based protocols do not need to exchange extra coordination messages, because they assume synchronized clocks and bounded message deliveries. Each process has to take a local checkpoint after expiry of its local timer [14], [15], [16]. However, in our algorithm, a process takes a checkpoint after expiry of its local timer if and only if it has sent at least one message in the current checkpoint interval. Therefore, the total number of checkpoints is very less compared to above mentioned algorithms.

Neves and Fuchs [3] proposed a checkpointing scheme that achieves global consistent checkpoint without additional messaging. The unique feature of the scheme is use of time to synchronize checkpoint creation [17]. They used time to indirectly coordinate the creation of recoverable consistent checkpoints. It requires that checkpoints be sent back only to home agents, which results in high failure-free overhead during checkpointing [18], [19]. However, our protocol does not have this limitation because each process takes its local checkpoint on the stable storage of its current local MSS in the protocol.

Neogy *et al*. [20], [21] proposed a synchronous checkpointing algorithm for distributed systems. Each process takes its checkpoint at predetermined time instants according to its own local clock. There must be some synchronization among various local clocks to make the checkpoints consistent. This problem is addressed by using extra messages for clock synchronization. The clock synchronization messages are used for synchronizing local clocks as well as checkpointing action. The authors made following assumptions in the system model:

- Logical clocks of the processes are at most D_{max} apart from each other. There exist a constant D_{max} such that at the k^{th} resynchronization interval (k>=0) for all correct clocks i and j, if the logical clocks of the processes P_i and P_j be denoted by C^k_i, and C^k_j, then $|C^k_i(t) - C^k_j(t)| <= D_{max}$
- Each process maintains a message log in the form: (message *id*, sender/receiver *id*)
- Since communication is synchronous, processes may be blocked for send/receive. Checkpointing instant may occur during the blocked period. Blocked processes take checkpoints after they unblock and hence, obviously record the event for which it was being blocked.
- An unblocked process takes its checkpoint when the checkpointing instant occurs. After taking the checkpoint it "freezes" itself (i.e. stops execution temporarily) and waits for the next clock resynchronization message. It appends its message log with the clock synchronization message so that processes slower than it can synchronize themselves.

This protocol has several advantages like there is no central checkpoint coordinator that poses a single point failure threat to the system. The overhead of coordinating messages is also absent. Synchronization among the local checkpoints has been brought about by the clock synchronization messages that have been utilized for this purpose. The method of checkpointing in blocked processes is not wastage of time because communication delay is finite. Since clock synchronization is an integral part of the above distributed system, "waiting" of a process, for the next clock synchronization message, does not also entail much time. As any global checkpoint is consistent, only one checkpoint needs to be stored in the stable storage. So, the system does not have to roll back more than once to restart from a previous consistent state in case recovery is required.

Limitations are that the system is synchronous hence the processes may be blocked at checkpointing instants. The sender process is blocked until acknowledgement is received or corresponding receive event is executed at the receiver process. Similarly, the receiver process is also blocked while waiting for the corresponding action at the sender's end. The blocked process takes checkpoint only after being unblocked. Our algorithm is nonblocking and uses timer to coordinate the process of checkpointing.

In [22], [23], each process takes checkpoint whenever its local timer expires. If a checkpoint initiator does not transitively depend on a process, the process does not have to take a checkpoint associated with the initiator. The result is that the number of checkpoints is minimized. The protocol is also nonblocking because the inconsistency between processes is avoided by piggybacking necessary information in each message. A timer has to be synchronized to avoid inconsistencies. This can be done using the piggybacked information. This protocol reduces the number of checkpoints compared to [14], [24], [3], [25], the traditional time-based protocols. It also makes use of the accurate timers in the MSS to adjust the timers in the MH, so that this protocol is well suited to mobile computing systems with MHs spread across a wide area network. The algorithm takes a number of soft checkpoints and computes dependency relationship using

extra control messages. Subsequently, on the basis of dependency relationship unnecessary soft checkpoints are discarded. However, in our algorithm a process takes checkpoints only if other processes are dependent on it, that is, it has sent at least one message to them in the current checkpoint interval. As the proposed algorithm takes only minimal checkpoints, there is no need to compute the dependency relation. Hence, our algorithm has less checkpointing overhead.

In [13], a two phase technology is used to handle potential inconsistency. Assume that maximum deviation that separates two timers, initiated in different nodes, is D and the checkpoint period is T. After n checkpoints have been created since the last timer re-synchronization, the maximum deviation of two processes is MD, where $MD=D+2\rho nT$. The inconsistency issues, that may arise, are messages sent before or after taking checkpoint MD time units. These messages can be in-transit or orphan messages. Assume, the maximum and minimum message propagation delays are t_{max} and t_{min}. The messages sent, $MD+t_{max}$ time units before taking checkpoint, may become in-transit messages and the messages sent, $MD-t_{min}$ (denoted as ED) time units after taking checkpoint, may become orphan messages.

In two phase technology two timers, *Timer_ckp* and *Timer_pmt*, are used to solve potential inconsistent issues. Whenever *Timer_ckp* expires, tentative checkpoint is taken and when *Timer_pmt* expires it converts tentative checkpoint into permanent checkpoint. *Timer_ckp* is set to T and *Timer_pmt* is set to $T+D+2n\rho T$, where n is checkpoint sequence number. The messages sent, in $MD+t_{max}$ time units before *Timer_ckp* expires, are logged by saving in *queue_in_transit* to avoid inconsistencies and the messages sent, in ED time after *Timer_ckp* expires, are sent with checkpoint sequence number (CSN), so that receiver compares and takes forced checkpoint decision depending on its CSN to avoid inconsistencies.

The protocol avoids all potential inconsistent issues that may arise in time-based algorithms [16], [3], [25]. The algorithm is also non-blocking. First tentative checkpoints are taken and these tentative checkpoints are later converted into permanent checkpoints. Hence, for each k^{th} checkpoint interval two checkpoint computations are required. However, our algorithm takes permanent checkpoints only, therefore, it is computationally more efficient.

B. Index-based Checkpointing Protocols

In index-based checkpointing protocols, each process takes checkpoints either at its own pace (basic checkpoints) or induced by some communication pattern (forced checkpoints) [7]. In Briatico *et al*. algorithm [26], the number of forced checkpoint induced by a basic one, in the worst case, is n-1, where n is the number of processes. Manivannan and Singhal [27] suggested an improvement over it. There is no reason to take a basic checkpoint if at least one forced checkpoint has been taken during the current checkpoint interval. However, the number of forced checkpoints remains at n-1, in the worst case. Baldoni *et al*. [7] presented a further improved index-based checkpointing algorithm for distributed systems that reduces the total number of worst case checkpoints while ensuring that each checkpoint belongs to at least one consistent global checkpoint. He argued that, due to different rate of basic checkpoint for different processes, the faster process forces the slower one to take more checkpoints. Hence, the number of forced checkpoint can be very high, resulting in large number of total checkpoints. Therefore, Baldoni's protocol does not increase the checkpoint sequence number, after taking basic checkpoints, unless the occurrence of a particular checkpoint and a communication pattern. The rationale behind his solution is that each time a basic checkpoint is taken without increasing the sequence number, it does not force checkpoints and this reduces the total number of checkpoints. In order to reduce the total number of checkpoints each application message is piggybacked with n+1 integers as control information, where n is the total number of processes. However, in our algorithm due to same checkpoint period for every process, the above mentioned problem does not arise. Therefore, not only the number of forced checkpoints is less, also, the size of control information is independent of the number of processes. The smaller control information results in lower message propagation delay.

In [5], the authors have defined and studied a family of communication-induced checkpointing protocols, where no local checkpoint is useless. A useless checkpoint is a local checkpoint that cannot be part of a consistent global checkpoint. However, in their protocols, the size of control messages is dependent on the number of processes. This problem has been addressed in [6]. Helary *et al*. [5] increased the number of forced checkpoints that are taken in order to eliminate the generation of useless local checkpoints. Tsai [6] further increased the number of forced checkpoints so that the size of control information piggybacked with each message becomes constant. In our algorithm, the size of control information is constant and the number of forced checkpoint taken is also less as every process takes checkpoint at a predefined rate which does not vary from process to process.

In distributed computation, processes take basic checkpoints in a different rate, some is faster, and some is slower. The result is two fold disadvantages. The slower processes, receiving large number of messages from the faster ones, have to take large number of forced checkpoints. Also, the recovery of slower processes, having too lower local checkpointing rate, results in the huge loss of computation. In order to avoid such situations, Luo *et al*. [28] presented an active approach by sending an extra message to the slower process to let it catch up. An extra synchronous message is sent from the faster process to the slower one, to let the slower process take a forced checkpoint with the large index. The message contains nothing but checkpoint index of the faster process. However, this problem does not arise in our algorithm because the algorithm is time-based and the checkpoint frequency for all processes is same.

The work closest to ours is Gupta *et al.*'s protocol [2]. His work is an improvement over mutable checkpoints [29]. It has reduced the number of checkpoints as compared to mutable checkpoints. It has the following good features:

1. Only those processes that have sent some message(s) after their last checkpoints, take checkpoints during checkpointing thereby reducing the number of checkpoints to be taken.
2. Reductions in the number of checkpoints help in the efficient use of the limited resources of mobile computing environment.
3. Uses minimum interaction (only once) between the initiator process and the system of n processes and there is no synchronization delay.
4. Uses very simple data structures, viz., three integer variables and one boolean variable per process.
5. Each process takes its checkpointing decision independently.

However, the algorithm has a limitation too. Consider a system of n process distributed system. Let, C_{cr} be the cost of sending a checkpoint request message from initiator to a single process. Hence, the checkpoint request cost, incurred by a single execution of the checkpointing algorithm, would be $(n\text{-}1)C_{cr}$. Thus, the checkpoint request cost, incurred by k executions of the checkpointing algorithm, would amount to $k(n\text{-}1)C_{cr}$. Therefore, the checkpoint request overhead, for applications involving large number of processes and running for longer durations, increases exponentially.

In the present work, we have attempted to eliminate above problem by using timer. It is a well-known fact that the use of timer eliminates extra coordination messages [10]. A process takes checkpoint whenever its local timer expires. Moreover, only those processes take checkpoint, after expiry of their local timer, who have sent at least one message in the current checkpoint interval. Therefore, the number of processes taking checkpoint and, subsequently, the total number of checkpoints is significantly reduced. In addition, the use of timer removes need of the initiator process for sending the checkpointing request messages.

VI. Conclusion

In this work we have developed an index-based algorithm which uses time-coordination for consistently checkpointing in mobile computing environments. The main features of our algorithm are: (1) it is non-blocking; (2) it is adaptive because it takes checkpointing decision on the basis of checkpoint sequence number; (3) it takes reduced number of checkpoints because a process does not take any temporary checkpoint and the process takes checkpoint if and only if some other process is dependent on it; (4) it doesn't require tracking and computation of dependency information; (5) it doesn't require any control message because it uses timer to indirectly coordinate the creation of consistent global checkpoints and the local timers are not synchronized through control messages but by piggybacking control information on application messages.

In time-based checkpointing protocols, there is no need to send extra coordination messages. However, they have to deal with the synchronization of timers. This class of protocols suits to the applications where processes have high message sending rate [13]. However, for the applications, where processes have low message sending rate, time-coordinated protocols may perform poorly. In our approach, timer synchronization is done through the control information, which is piggybacked on application messages. Hence, in case, the application has low message sending rate, timers would not be synchronized frequently. Therefore, in case of a process failure, the system has to rollback, to a comparatively older state, loosing a significant amount of computation. The problem is being postponed for a future research work.

References

[1] K. M. Chandy and L. Lamport, "Distributed snapshots: determining global states of distributed systems," *ACM Transactions on Computer Systems*, 1985, February, vol. 3, no. 1, pp. 63–75.

[2] B. Gupta, S. Rahimi, R. Rias, and G. Bangalore, "A low-overhead non-block checkpointing algorithm for mobile computing environment," in *Lecture Notes in Computer Science 3947*, *GPC 2006*, Springer-Verlag, 2006, pp. 597–608.

[3] N. Neves and W. Fuchs, "Adaptive recovery for mobile environments," in *Proc. IEEE High-Assurance Systems Engineering Workshop*, October 21-22, 1996, pp.134–141. Also in *Communications of the ACM*, 1997, January, vol. 40, no. 1, pp. 68–74.

[4] B. Randell, "System structure for software fault-tolerance," *IEEE Transactions on Software Engineering*, 1975, June, vol. 1, no. 2, pp. 220–232.

[5] J. Helary, A. Mostefaoui, R. Netzer, and M. Raynal, "Communication-based prevention of useless checkpoints in distributed computations," *Distributed Computing*, 2000, January, vol. 13, no. 1, pp. 29–43.

[6] J. Tsai, "An efficient index-based checkpointing protocol with constant-size control information on messages," *IEEE Transactions on Dependable and Secure Computing*, 2005, Oct-Dec, vol. 2, no. 4, pp. 278–296.

[7] R. Baldoni, F. Quaglia, and P. Fornara, "An index-based checkpointing algorithm for autonomous distributed systems," in *Proc. 16th IEEE Symposium on Reliable Distributed Systems*, October 22-24, 1997, pp.27–34. Also in *IEEE Transactions on Parallel and Distributed Systems,* 1999, February, vol. 10, no. 2, pp. 181–192.

[8] L. Lamport, "Time, clocks and the ordering of events in a distributed system," *Communications of the ACM*, 1978, July, vol. 21, no. 7, pp. 558–565.

[9] L. Alvisi, E. Elnozahy, S. Rao, S. A. Husain, and A. D. Mel, "An analysis of communication-induced checkpointing," in *Proc. IEEE Fault-Tolerant Computing Symposium*, June 15-18, 1999, pp. 242–249.

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

[11] N. Vaidya, "Another two-level failure recovery scheme: performance impact of checkpoint placement and checkpoint latency," Tech. Rep. 94-068 (revised), Texas A&M University, January, 1995.

[12] N. Vaidya, "On checkpoint latency," in *Proc. Pacific Rim International Symposium on Fault-Tolerant Systems*, December, 1995, pp. 60–65.

[13] M. Chaoguang, Z. Yunlong, and Y. Wenbin, "A two-phase time-based consistent checkpointing strategy," in *Proc. ITNG'06 3rd IEEE International Conference on Information Technology: New Generations*, April 10-12, 2006, pp. 518–523.

[14] Z. Tong, R. Kain, and W. Tsai, "A low overhead checkpointing and rollback recovery scheme for distributed systems," in *Proc. 8th IEEE Symposium on Reliable distributed systems*, October 10-12, 1989, pp. 12–20.

[15] F. Cristian and F. Jahanian, "A timestamp-based checkpointing protocol for long-lived distributed computation," in *Proc. 10th IEEE Symposium on Reliable Distributed Systems*, 30 Sep-02 Oct, 1991, pp. 12–20.
[16] N. Neves and W. Fuchs, "Using time to improve the performance of coordinated checkpointing," in *Proc. 2nd IEEE International Computer Performance and Dependability Symposium*, September 4-6, 1996, pp. 282–291.
[17] S. George, I. Chen, and Y. Jin, "Movement-based checkpointing and logging for recovery in mobile computing systems," in *Proc. MobiDE'06 5th ACM International Workshop on Data Engineering for Wireless and Mobile Access*, June, 2006, pp. 51–58.
[18] J. Ahn and C. Hwang, "Low-cost fault-tolerance for mobile nodes in mobile IP based systems," in *Proc. IEEE International Conference on Distributed Computing Systems Workshop*, April 16-19, 2001, pp. 508–513.
[19] J. Ahn, S. Min, and C. Hwang, "A causal message logging protocol for mobile nodes in mobile computing systems," *Future Generation Computer Systems*, 2004, May, vol. 20, no. 4, pp. 663–686.
[20] S. Neogy, A. Sinha, and P. Das, "Checkpoint processing in distributed systems software using synchronized clocks," in *Proc. ITCC 2001 IEEE International Conference on Information Technology: Coding and Computing*, April 2-4, 2001, pp. 555–559.
[21] S. Neogy, A. Sinha, and P. Das, "Distributed checkpointing using synchronized clocks," in *Proc. COMPSAC'02 26th IEEE Annual International Computer Software and Applications Conference*, 2002, pp. 199–206.
[22] C. Lin, S. Wang, S. Kuo, and I. Chen, "A low overhead checkpointing protocol for mobile computing systems," in *Proc. PRDC'02 Pacific Rim International Symposium on Dependable Computing*, December 16-18, 2002, pp. 37–44.
[23] C. Lin, S. Wang, and S. Kuo, "An efficient time-based checkpointing protocol for mobile computing systems over wide area networks," in *Lecture Notes in Computer Science 2400*, *Euro-Par 2002*, Springer-Verlag, 2002, pp. 978–982. Also in *Mobile Networks and Applications*, 2003, vo. 8, no. 6, pp. 687–697.
[24] Z. Tong, R. Kain, and W. Tsai, "Rollback recovery in distributed systems using loosely synchronized clocks," *IEEE Transactions on Parallel and Distributed Systems*, 1992, March, vol. 3, no. 2, pp, 246–251.
[25] N. Neves and W. Fuchs, "Coordinated checkpointing without direct coordination," in *Proc. IPDS'98 IEEE International Computer Performance and Dependability Symposium*, September 7-9, 1998, pp. 23–31.
[26] D. Briatico, A. Ciuffoletti, and L. Simoncini, "A distributed domino-effect free recovery algorithm," in *Proc. 4th IEEE International Symposium on Reliability in Distributed Software and Database Systems*, October, 1984, pp. 207–215.
[27] D. Manivannan and M. Singhal, "A low-overhead recovery technique using quasi synchronous checkpointing," in *Proc. 16th IEEE International Conference on Distributed Computing Systems*, May, 1996, pp. 100–107.
[28] Y. Luo, Y. Min, and D. Zhang, "An improved scheme of index-based checkpointing," in *Proc. 11th IEEE Pacific Rim International Symposium on Dependable Computing*, December 12-14, 2005, Page(s): 5 pp.
[29] G. Cao and M. Singhal, "Mutable checkpoints: a new checkpointing approach for mobile computing systems," *IEEE Transactions on Parallel and Distributed Systems*, 2001, February, vol. 12, no. 2, pp. 157-172.