# A New Efficient Checkpointing Algorithm for Distributed Mobile Computing

**Houssem Mansouri[1], Nadjib Badache[2], Makhlouf Aliouat[3], Al-Sakib Khan Pathan[4]**

*[1]Department of Computer Science, Faculty of Exact Sciences, University of Bejaia, 06000, Bejaia, Algeria, (e-mail: mansouri_houssem@yahoo.fr).*
*[2]Research Center on Scientific and Technical Information CERIST, Ben-Aknoun, Algiers, 16000, Algeria, (e-mail: badache@mail.cerist.dz).*
*[3]Laboratory of Networks and Distributed Systems, Computer Science Department, University of Ferhat Abbas Sétif1, Sétif, 19000, Algeria, (e-mail: aliouat_m@yahoo.fr).*
*[4]Department of Computer Science, International Islamic University Malaysia, Kuala Lumpur, 53100, Malaysia, (e-mail: sakib@iium.edu.my).*

**Abstract:** Mobile networks have been quickly adopted by many companies and individuals. However, multiple factors such as mobility and limited resources often constrain availability and thus cause instability of the wireless environment. Such instability poses serious challenge for fault tolerant distributed mobile applications. Therefore, the classical *checkpointing* techniques, which make the applications more *failure-resistant*, are not always compatible with the mobile context. In fact, it is necessary now to think about other techniques or at least adapt those to devise effective and well suited techniques for the mobile environment. Considering this issue, the contribution in this paper is a proposal of a new *checkpointing* algorithm suitable for mobile computing systems. This algorithm is characterized by its efficiency and optimization in terms of incurred time-space overhead during *checkpointing* process and normal application running period.

*Keywords:* Distributed mobile computing, Fault tolerance, Consistent global state, Coordinated *checkpointing*.

## 1. INTRODUCTION

The increasing development of wireless communication technology during the last decade has triggered the rapid growth of mobile networks. Due to the high flexibility of use, the latest technologies have been quickly adopted by companies and individuals all over the globe. In spite of all the advancements and quick adoptions today, a full-fledged ubiquitous Internet network that could widely exploit the capabilities of the mobile networks, is still not possible to build. This is because of the high cost associated with infrastructure building and limited transmission range. However, it is expected that these constraints would tend to decrease as the wireless technologies progress – perhaps, in the near future, all these obstacles would be abolished and Internet access will be possible from anywhere via usual mobile networks.

Nevertheless, the difficulties, e.g., unavailability, instability, that are seen today basically come from the inherent characteristics of mobile networks and wireless communications. These negative aspects should be mitigated as much as possible so that the Quality of Service (QoS) offered to users may be satisfactory. As services delivered in a mobile environment are in fact, the extensions of services provided by the classic distributed systems, numerous algorithms or usual algorithms which carry out these services have to be modified or rethought in a better way to adapt them to the constrained mobile network environment.

Among various algorithms to be adjusted to meet the requirements of mobile applications, we mention in particular the *checkpointing* algorithms that enable these applications and systems to have fault tolerance capabilities. If the appropriate adjustments could be done, indeed, mobile distributed systems will then be capable of running critical distributed applications within prescribed time (in accordance with their mission specifications).

To be suitably adapted to this new environment, *checkpointing* algorithms have to take into account the following issues (Badrinath et al., 1993, 1994; Acharya and Badrinath, 1994; Forman and Zahorjan, 1994):

- **Low bandwidth:** it is important that a *checkpointing* algorithm generates a minimum of coordinating messages, and minimizes the most likely extra piggybacked information.
- **Limitation of storage space:** the lack of disk space on Mobile Host (MH) implies that checkpoints will be transferred in the Mobile Support Stations, which have sufficient storage size.
- **Handling of mobility:** frequent movement and disconnection of the network put an obstacle in the synchronization process or the coordination of the MHs during the process of *checkpointing* or recovering.
- **Limited energy source:** conservation and optimization of energy consumption is a paramount importance. Any use of

an MH has to take into account the weak life expectancy of the energy source.

After this introduction, the rest of the sections are organized as follows: Section 2 talks about the system model and assumptions, Section 3 introduces various facets of the *checkpointing* algorithm, Section 4 talks about the phases of the algorithm, Section 5 presents the *pseudocode* of the algorithm, followed by the proof of correctness in Section 6. Performance evaluation is presented in Section 7 before concluding the paper in Section 8 with future research directions.

## 2. SYSTEM MODEL AND ASSUMPTIONS

The system model shown in Figure 1, which has been taken into consideration by various other works (Acharya and Badrinath, 1994; Dey and Biswas, 2012; Kiehn et al., 2014), consists of a set of wireless Mobile Hosts (MHs) and fixed base stations called Mobile Support Station (MSSs) attached to MHs. The static MSS is capable to provide various services to support MHs. The geographic region covered by MSS range transmissions is called a cellule (*Cell*). Bidirectional communication between an MH and MSS is established via a wireless link with message sending FIFO (First-In, First-Out) strategy. Any communication between any two MHs: $MH_i$ and $MH_j$ is accomplished only via corresponding $MSS_i$ and $MSS_j$ - no direct communication between MHs is allowed. A high speed wired communication link is assumed to connect any two MSSs. Message transmissions through the wired and wireless links take an unpredictable, but finite amount of time.

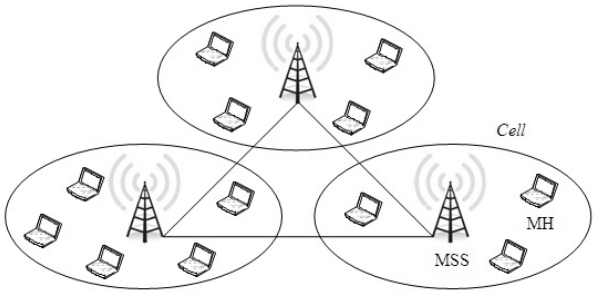



Fig. 1. Mobile environment.

An application running in a mobile computing system is assumed to be composed of $n$ concurrent processes, which collectively perform a distributed computation. It is assumed that each process runs on a distinct MH. The computation is assumed to be conformed to a piece-wise deterministic model in which a process always produces the same sequence of *states* in its execution for the same sequence of message-receiving events. In this paper, we consider the transient-failure type, that is, a process is unlikely to fail again at the same execution point after it recovers from a failure. Fail-silent mode for processes is assumed, so when a process fails, it immediately stops the execution and does not perform any malicious action. This fail-silent property (Schlichting and Schneider, 1983) is important for considering that messages sent prior to a failure, are correct. Otherwise, we have to take into account the hard problem of checkpoints' trustworthiness which can lead to propagate state restoration in a recovery operation similar to a domino-effect mechanism (Elnozahy et al., 2002).

We also assume the presence of reliable message delivery, that is, there is no message loss or alteration during normal computation. Stable storage is assumed to survive in case of MH or MSS failure while volatile memory loses its content. The degree of fault tolerance in the wired network and the computation capabilities of the MSSs are supposed to be high.

## 3. CHECKPOINTING ALGORITHM

A *checkpointing* algorithm suitable for mobile environment has to be in accordance with the following two essential aspects (Badrinath et al., 1993, 1994; Acharya and Badrinath, 1994):

1. Taking into account the constrained mobile environment, notably the low bandwidth, the frequent handoff of mobile hosts, and their limited storage space, MHs must be given the option to freely set their own checkpoints. Because, only MHs have sufficient knowledge when it is favourable to do that. Therefore, the MHs may take advantage of propitious situations to save checkpoints, e.g., when the size of checkpoint is adequate, the battery level is high or a handoff is soon expected, etc. However, this freedom accorded to MHs may cause a well known domino-effect problem. On the other hand, the large resource availability in the MSSs and wired network makes the restrictions in *checkpointing* less severe and recommends the use of coordinated *checkpointing* technique. In order to comply with mobile context specifications and getting to implement pertinent recovery capabilities, hybrid *checkpointing* algorithms are needed.

2. For new constraints, new directions of investigation are needed. Indeed, in wired networks, *checkpointing* algorithms have no need to handle handover situations especially making difference between a long handoff and an MH failure. In the case of failure caused by depletion of the battery, we may have to face two situations. The first one is the definitive shutdown of the MH making it impossible to complete the application. This is the common situation reported in all existing *checkpointing* algorithms until now. The second one is the one that tries to overcome this problem by replicating (in MSS) activities taking place in MH in order to recover energy source from changing battery. This difficult alternative (but which is also convenient to insure mobile distributed applications' non-stop property) is under our current investigation.

### *3.1 Technique of checkpointing used*

Our algorithm is based on the following strategies:

1. Coordinated *checkpointing* technique between MSSs to save a consistent global state and to avoid orphan messages (Chandy and Lamport, 1985; Netzer ans Xu, 1995; Elnozahy et al., 2002). The use of this technique is aimed at getting simplicity in the implementation of the algorithm and the insurance to always have a consistent global state available during the recovery operation. Indeed, when a local

checkpoint is saved, the algorithm guarantees that this checkpoint is part of a global consistent state. Thus, MSSs have to cooperate with the object to keep only a single checkpoint for every MH. The storage space required for *checkpointing* is then optimized. Once failure occurs, each MH concerned with recovery, simply has to retry execution from its last permanent checkpoint.

2. Coordination is not *blocking* (Cao and Singhal, 2001; Kanmani et al., 2007; Awasthi et al., 2014), that is, there is no need to momentarily stop the computation of the mobile distributed application during the execution of the *checkpointing* process. This significantly increases application performance, because the algorithm is performed in a transparent way. However, when a mobile host $MH_i$ sends a message *m* to its partner $MH_j$ via its associated $MSS_i$, $MSS_i$ piggybacks with the message *m* the current value of $MH_i$'s checkpoint number: *Ckpti*. When the $MSS_j$ station associated to $MH_j$ receives the message *m*, *m* is relayed (by $MSS_j$) to $MH_j$ if the $MH_j$'s checkpoint number *Ckptj* is superior or equal to the checkpoint number *Ckpti* of the sender $MH_i$. Otherwise, $MSS_j$ sends first a *checkpointing* request to $MH_j$ before sending it the message *m*, and then updates the checkpoint number, *Ckptj*.

3. Coordination is maintained minimum (Cao and Singhal, 2001), that is, the algorithm does not force all the mobile hosts to take checkpoints. Only hosts having communicated with the initiator of the algorithm directly or indirectly after the last taken checkpoint have to save a checkpoint. The algorithm includes two phases: The first consists of an identification of all the MHs causally depending on the MH initiator (since its last checkpoint) which sends them a *checkpointing* request. On reception of this last one, every MH identifies all the MHs with which it had to communicate since the last checkpoint and sends them a *checkpointing* request, and so on until that there is no more MH to be identified. In the second phase, all the MHs identified during the first phase, go for setting their permanent checkpoint. Hence, the result obtained is a consistent global state with the cooperation of an optimal number of MHs.

4. Our algorithm uses the message logging technique (Park et al., 2002) to replay messages intended for a mobile site MH during its disconnection.

### *3.2 Abbreviations and data structure used*

**Abbreviations:** To simplify and to make our presentation as clear as possible, we use the following abbreviations:

*MovReq*: moving request.
*DiscReq*: disconnection request.
*RecReq*: reconnection request.
*CkptReq* : *checkpointing* request.
*ValidReq* : validation request.
*TransInformReq*: transmission information request.

**Data structure used:**

**For every mobile host $MH_i$ we have:**

*Ckpti*: indicates the number of the last checkpoint recorded in $MH_i$.
*DepdSitesi*: set of MHs which have a causal dependence relation with $MH_i$.
*Logi* : set of computation messages sent to $MH_i$ during their disconnection.
*DiscCkpti*: used to save the last disconnection checkpoint of $MH_i$.
*TempCkpti*: used to save the last temporary checkpoint of $MH_i$.
*PermCkpti*: used to save the last permanent checkpoint of $MH_i$.

**For every base station *MSSn* we have:**

*PresSitesn*: set of MHs present in $cell_n$.
*DiscSitesn*: set of MHs disconnected from $cell_n$.
*GlobSetn*: set of MHs involved in checkpoint recording.
*TempInSetn*: set of MHs identifiers internal to $cell_n$.
*TempExtSetn*: set of MHs identifiers external to $cell_n$.
*Ckptn*: variable for saving the number of the new checkpoints.
*InitMSS*: used to save the identity of the initiator base station.
*Term*: used to detect algorithm termination.

### *3.3 Handling of mobile environment characteristics*

A *checkpointing* algorithm suitably adapted to the mobile context has to satisfy several criteria to integrate the specific characteristics of this environment and their influences on the distributed computation. Among these is "*Mobile host mobility processing*". When moving from place to place, a mobile host $MH_i$ has to inform its basic station $MSS_i$ for all its movements. Hence, when $MH_i$ moves from a $Cell_i$ (supported by $MSS_i$) to another $cell_j$ (supported by $MSS_j$), $MH_i$ has to inform about it to the former station, $MSS_i$ and the new one, $MSS_j$. However, it is necessary to make sure that the data structure of the mobile host (*Ckpti, DepdSitesi, Logi, DiscCkpti, TempCkpti, PermCkpti*) is always available at the current base station.

**A. Mobile host movement processing:** As shown in Figure 2, at the time of movement, the mobile host $MH_i$ sends a movement request to the new base station $MSS_j$. Upon reception of the request, $MSS_j$ sends a request to the former base station $MSS_i$ to get information concerning the mobile site $MH_i$. $MSS_i$ sends then to $MSS_j$ what is requested.

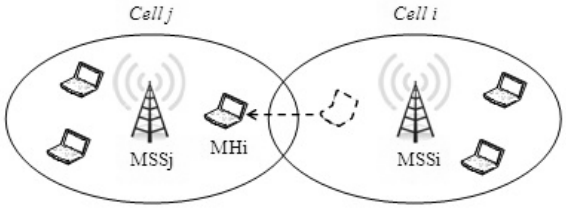



Fig. 2. MH Movement.

**Routine of movement handling**
**-Role of the mobile host $MH_i$:**
At the time of moving:
***Send*** *MovReq ($MH_i$, $MSS_i$);*
**-Role of the new base station $MSS_j$:**
Upon receiving moving request from $MH_i$:
***Receive*** *MovReq ($MH_i$, $MSS_i$);*
*PresSitesj := PresSitesj + $MH_i$;*
***Send*** *TransInformReq ($MSS_i$, $MSS_j$, $MH_i$);*

Upon receiving requested information from $MSS_i$:
***Receive*** *Information ($MSS_j$, $MH_i$, Ckpti, DepdSitesi, Logi, DiscCkpti, TempCkpti, PermCkpti);*
***Save*** *(Ckpti, DepdSitesi, DiscCkpti, TempCkpti, PermCkpti);*

**-Role of the former base station *MSSi*:**

Upon receiving information transmission request from $MSS_j$:
***Receive*** *TransInformReq ($MSS_i$, $MSS_j$, $MH_i$);*
***Send*** *Information ($MSS_j$, $MH_i$, Ckpti, DepdSitesi, Logi, DiscCkpti, TempCkpti, PermCkpti);*
*PresSitesi := PresSitesi - $MH_i$;*

**B. Mobile host disconnection and reconnection processing:** As shown in Figure 3, the mobile host $MH_i$ records a checkpoint (*DiscCkpt*) before its disconnection and sends it to the former base station $MSS_i$. The mobile host $MH_i$ sends then a request of reconnection (*RecReq*) to the new base station $MSS_j$. The new base station $MSS_j$ sends a request to the former base station $MSS_i$ to get back information concerning the mobile host $MH_i$. The former base station $MSS_i$ sends to the new base station $MSS_j$, the information relating to the mobile host, $MH_i$.

The new base station $MSS_j$ sends the disconnection checkpoint (*DiscCkpt*) got back from $MSS_i$, to the mobile host, $MH_i$. $MH_i$ has also to get messages (sent to it during disconnection) from $MSS_i$, according to the order of their previous reception. The mobile host $MH_i$ loads then its state with the received checkpoint and handles messages.

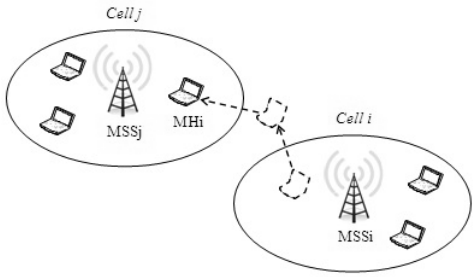



Fig. 3. Disconnection/Reconnection scenario.

**Routine of disconnection handling**

**-Role of the mobile host *MH$_i$*:**

Before the disconnection:
***Record*** *(DiscCkpti);*
***Send*** *DiscReq (MHi, DiscCkpti);*

**-Role of the former base station *MSS$_i$*:**

Upon receiving disconnction request from $MH_i$:
***Receive*** *DiscReq ($MH_i$, DiscCkpti);*
*DiscSitesi := DiscSitesi + $MH_i$;*
***Until (Receive*** *TransInformReq ($MSS_i$, $MSS_j$, $MH_i$)) **Do***
Upon reception of application message from $MH_n$ for $MH_i$:
***Receive*** *Message ($MH_i$, $MH_n$, Message);*
*Logi := Logi + ($MH_i$, $MH_n$, Message);*
***End***

**Routine of reconnection handling**

**-Role of the mobile host *MH$_i$*:**

At the time of reconnection to $MSS_j$:
***Send*** *RecReq ($MH_i$, $MSS_i$);*
Upon receiving disconnection checkpoint from $MSS_j$:
***Receive*** *Message ($MH_i$, DiscCkpti);*
***Load*** *(DiscCkpti);*
Upon receiving application message from $MH_n$:
***Receive*** *Message ($MH_i$, $MH_n$, Message);*
***Handle*** *(Message);*

**-Role of the new base station *MSSj*:**

Upon receiving reconnection request from $MH_i$:
***Receive*** *RecReq ($MH_i$, $MSS_i$);*
*PresSitesj := PresSitesj + $MH_i$;*
***Send*** *TransInformReq ($MSS_i$, $MSS_j$, $MH_i$);*
Upon receiving requested information from $MSS_i$:
***Receive*** *Information ($MSS_j$, $MH_i$, Ckpti, DepdSitesi, Logi, DiscCkpti, TempCkpti, PermCkpti);*
***Save*** *(Ckpti, DepdSitesi, DiscCkpti, TempCkpti, PermCkpti);*
***Send*** *Message ($MH_i$, DiscCkpti);*
***For All*** *($MH_i$, $MH_n$, Message) $\in$ Logi **Do***
***Send*** *Message ($MH_i$, $MH_n$, Message);*
*DepdSitesi := DepdSitesi + $MH_n$;*
***End***
*Logi := $\varnothing$;*
*DiscCkpti := $\varnothing$;*

**-Role of the former base station *MSS$_i$*:**

Upon receiving information transmission request from *MSSj*:
***Receive*** *TransInformReq ($MSS_i$, $MSS_j$, $MH_i$);*
***Send*** *Information ($MSS_j$, $MH_i$, Ckpti, DepdSitesi, Logi, DiscCkpti, TempCkpti, PermCkpti);*
*PresSitesi := PresSitesi - $MH_i$;*
*DiscSitesi := DiscSitesi - $MH_i$;*

## 4. PHASES OF THE ALGORITHM

The algorithm works in two phases: *(i)* The phase of the creation of the temporary checkpoints, and *(ii)* the phase of transformation of the temporary checkpoints to permanent ones.

**1. First phase:** When a mobile host, $MH_r$ runs the *checkpointing* algorithm (it is assumed that there is only a single instance of *checkpointing* process at a time), it records a temporary checkpoint and sends a *checkpointing* request to its base station $MSS_n$. Upon reception of the request, $MSS_n$ determines the identifier set *DepdSitesr*. $MSS_n$ then broadcasts *checkpointing* request to all the base stations (included itself). This request piggybacks the set *DepdSitesr* determined before.

During all the execution of the algorithm, if a base station $MSS_i$ receives a *checkpointing* request *CkptReq (TempExtSetx, Value)*, then $MSS_i$ in turn broadcasts a *checkpointing* request to the mobile hosts, $MH_i \in TempExtSetx$ of its own cellule, $MH_i \in Cell_i$ concerned with the *checkpointing* process. $MSS_i$ also determines the identifiers of the mobile hosts directly dependent on these mobile hosts (which are also indirectly dependent on the initiator host $MH_r$). If the set so determined is empty, then the base station $MSS_i$ sends an acknowledgment to the initiator base station $MSS_n$; otherwise, it broadcasts this set to all other base stations, and so on.

**2. Second phase:** When the initiator base station, $MSS_n$ receives an answer from all the base stations $MSS_i$ ($i \in [1,$

*m*]), then it broadcasts a request of validation to all the base stations. Every base station, having received the validation request, has to broadcast in turn, this request in its own cellule. So, every mobile host in the cellule, having received this request, has to send its temporary checkpoint (if there is any) to the base station to make it permanent and save it in the stable storage.

## 5. THE ALGORITHM PSEUDOCODE

**Initialization:**
For every mobile host *MHi*:
***Record*** *(PermCkpti);*
***Send*** *Message ($MH_i$, PermCkpti)*
For every base station $MSS_j$:
*PresSitesj := {$MH_i$ / $MH_i \in Cell_j$} ;*
*DiscSitesj := ∅;*
*GlobSetj := ∅;*
*TempInSetj := ∅;*
*TempExtSetj := ∅;*
*InitMSS := ∅;*
*Term := 0 ;*
***For All*** *($MH_i \in$ PresSitesj)* ***Do***
***Receive*** *Message ($MH_i$, PermCkpti)*
***Save*** *(PermCkpti)*
*Ckpti := 1 ;*
*DepdSitesi := ∅;*
*Logi := ∅;*
*DiscCkpti := ∅;*
*TempCkpti := ∅;*
***End***
*Ckptj := 1 ;*
**Initiation of the algorithm**
**-Role of the initiator mobile host $MH_r$:**
***Record*** *(TempCkptr);*
***Send*** *CkptReq ($MH_r$);*
**-Role of the initiator base station $MSS_n$:**
Upon receiving *checkpointing* request from the initiator mobile host $MH_r$:
***Receive*** *CkptReq ($MH_r$);*
*Ckptr := Ckptr + 1;*
*Ckptn := Ckptr;*
*TempExtSetn := DepdSitesr;*
***Send*** *CkptReq ($MSS_n$, Ckptn, TempExtSetn, 1/m);*
*GlobSetn := TempExtSetn;*
Upon receiving answer from a base station:
***Receive*** *answer ($MSS_n$, Value);*
*Term := Term + Value;*
***If*** *(Term = 1)* ***Then***
***Send*** *ValidReq($MSS_n$);*
***EndIf***
**During the execution of the algorithm:**
**-Role of every base station $MSS_i$:**
Upon receiving *checkpointing* request from a base station $MSS_x$:
***If Receive*** *CkptReq ($MSS_x$, Ckptx, TempExtSetx, Value)* ***Then***
*InitMSS := $MSS_x$;*
*Ckpti := Ckptx;*
***Else***
***Receive*** *CkptReq (TempExtSetx, Value);*
***Endif***
*TempInSeti := {$MH_p$ / $MH_p \in$ TempExtSetx ∧ $MH_p \in$ PresSitesi ∧ $MH_p \notin$ GlobSeti};*
*GlobSeti := GlobSeti ∪ TempExtSetx;*
***For All*** *($MH_p \in$ TempInSeti)* ***Do***
***If*** *($MH_p \in$ DiscSitesi)* ***Then***
*TempCkptp := DiscCkptp;*
***Else***
***Send*** *CkptReq ($MH_p$);*
***Endif***
*Ckptp := Ckpti;*
***Endfor***
*TempExtSeti := {$MH_t$ / $MH_t \in$ DepdSitesp / $MH_p \in$ TempInSeti};*
***If*** *(TempExtSeti = Ø)* ***Then***
***Send*** *answer (InitMSS, Value);*
***Else***
***Send*** *CkptReq (TempExtSeti, Value/m);*
***Endif***
For reception of a validation request from $MSS_n$:
***Receive*** *ValidReq ($MSS_n$);*
***For All*** *($MH_i \in$ GlobSeti ∧ $MH_i \in$ PresSitesi)* ***Do***
***If*** *($MH_i \in$ DiscSitesi)* ***Then***
*PermCkpti := TempCkpti ;*
***Else***
***Send*** *ValidReq ($MH_i$);*
***Endif***
***Endfor***
Upon receiving temporary checkpoint from $MH_j$:
***Receive*** *Message ($MH_i$, TempCkpti);*
*PermCkpti := TempCkpti ;*
**-Role of every mobile host $MH_i$:**
Upon receiving *checkpointing* request:
***Receive*** *CkptReq ($MH_i$);*
***Record*** *(TempCkpti);*
Upon receiving validation request:
***Receive*** *ValidReq ($MH_i$);*
***Send*** *Message ($MH_i$, TempCkpti);*

During the time of execution of the algorithm, to avoid blocking of the execution of mobile distributed application, if a base station $MSS_i$ receives an application message from $MH_i$ for $MH_j$, then $MSS_i$ piggybacks with the message the current value of $MH_i$'s checkpoint number: *Ckpti*. When the $MSS_j$ station associated to $MH_j$ receives the message, it is relayed (by $MSS_j$) to $MH_j$ if the $MH_j$'s checkpoint number *Ckptj* is superior or equal to checkpoint number *Ckpti* of the sender $MH_i$. Otherwise, $MSS_j$ sends first a *checkpointing* request to $MH_j$ before sending it the message, and then, updates the checkpoint number, *Ckptj*.

**-Role of every base station $MSS_i$:**
Upon receiving application message from $MH_i$ ($MH_i \in Cell_i$) for $MH_j$:
***Receive*** *Message ($MH_j$, $MH_i$, Message);*
***Send*** *Message ($MH_j$, $MH_i$, Ckpti, Message);*
Upon receiving application message from $MH_i$ for $MH_j$ ($MH_j \in Cell_j$):

***Receive*** *Message ($MH_j$, $MH_i$, Ckpti, Message);*
***If*** *(Ckptj < Ckpti)* ***Then***
***Send*** *CkptReq ($MH_j$);*
*Ckptj := Ckpti;*
*TempExtSeti := TempExtSeti + {$MH_t$ / $MH_t \in$ DepdSitesj}*
***Endif***
***Send*** *Message ($MH_j$, $MH_i$, Message);*
*DepdSitesj := DepdSitesj + $MH_i$;*

**-Role of every mobile host $MH_i$:**

For sending an application message to $MH_j$:
***Send*** *Message ($MH_j$, $MH_i$, Message);*

Upon receiving application message from $MH_j$:
***Receive*** *Message ($MH_i$, $MH_j$, Message);*
***Handle*** *(Message);*

## 6. PROOF OF CORRECTNESS

***Lemma 1*:** If a mobile host $MH_i$ initiates the *checkpointing* algorithm, then any other mobile host $MH_j$ directly or indirectly depends on $MH_i$ since its last checkpoint has to record a checkpoint.

**Correctness proof:** Let us demonstrate with some cases. If a mobile host $MH_i$ records a checkpoint, then any other mobile host $MH_j$, such as $MH_j \in DepdSitesi$ has to take a checkpoint.

**Case 1:** if a mobile host $MH_r$ (initiator of the algorithm) sends a *checkpointing* request to its base station $MSS_n$, $MSS_n$ sends in turn, this request to all the base stations with the set *TempExtSetn* of the identifiers of all the mobile hosts $MH_j$ such as $MH_j \in DepdSitesr$.

With the reception of this request, every base station $MSS_i$ acts as follows:

1. Broadcasts the request to all mobile hosts in set *TempExtSetn* which belong to the $Cell_i$. With the reception of a *checkpointing* request, every mobile host records a temporary checkpoint.
2. If there is a mobile host $MH_j$ such that $MH_j \in TempExtSetn \wedge MH_j \in DiscSitesi$, then, its base station $MSS_i$ transforms its disconnection checkpoint *DiscCkptj* into a temporary checkpoint *TempCkptj*.

As at any time, every mobile host is either connected to one of the base stations (or disconnected), if a mobile host runs the *checkpointing* algorithm, any other mobile host which directly depends on it (since it is the last checkpoint) has to establish a temporary checkpoint.

**Case 2:** if a mobile host $MH_i$ is not the initiator of the algorithm and receives *checkpointing* request from its base station $MSS_i$, then $MSS_i$ serves all the mobile hosts which are directly dependent on $MH_i$, and sends a *checkpointing* request to all the base stations (included itself) piggybacking the *TempExtSeti* set. After receiving this request, every base station $MSS_j$ acts as follows:

1. If one of these determined hosts belongs to the set *GlobSetj*, then this host has already recorded a temporary checkpoint, either in this station or in the other one.
2. If there is a mobile host $MH_k \in TempExtSeti \wedge MH_k \in DiscSitesj$, then the base station changes the disconnection checkpoint to a temporary one for this mobile host $MH_k$.
3. Broadcasts the request to all mobile hosts in set *TempExtSeti* which belong to the $Cell_j$. After receiving a *checkpointing* request, every mobile host records a temporary checkpoint.

At any instance of time, every mobile host is either connected to one of the base stations and it has previously recorded a checkpoint, or it is reconnected to one of the base stations and it did not first take a checkpoint, or it is not connected at all. Hence, if a mobile host records a checkpoint, any other mobile host would directly depend on it since it is the last host which recorded a checkpoint.

From Case 1 and Case 2, we can conclude in formal terms that:

If a mobile host $MH_i$ records a checkpoint, any other mobile host $MH_j$ such as $MH_j \in DepdSitesi$ has to record a checkpoint. Because of transitive relation dependency, if a mobile host $MH_i$ runs the *checkpointing* algorithm, any other mobile host $MH_j$ would directly or indirectly depend on it since the last checkpoint, has also to record a checkpoint.

***Lemma 2:*** The algorithm is convergent, that is it completes its execution within the prescribed amount of time.

**Correctness proof:** We defined the following predicates:

- ***Portion (Term, $MSS_n$)*:** the portion of *Term* actually saved in the initiator base station $MSS_n$.
- ***Portion (Term, CkptReq)*:** the portion of *Term* piggybacked in a *checkpointing* request which is in transit.
- ***Portion (Term, Answer)*:** the portion of *Term* piggybacked by an answer request in transit.

At the initialization phase of the algorithm, we have: *Portion (Term, MSSn)* = 0. If we can prove that after a finite time of the algorithm execution starting point, we get: *Portion (Term, MSSn)* = 1, we can conclude that the algorithm completes its execution in time. (Huang, 89).

We have to verify the following equation through different periods of the *checkpointing* algorithm execution:

$$Portion\ (Term, MSS_n) + \sum Portion\ (Term, CkptReq) + \sum Portion\ (Term, Answer) = 1 \quad (1)$$

**Period 1:** the initiator base station broadcasts a *checkpointing* request to all other base stations with the value $1/m$ for each of it:
$Portion\ (Term, MSS_n) = 0$;
$Portion\ (Term, CkptReq) = 1/m \Rightarrow \sum Portion\ (Term, CkptReq) = 1$;
$\sum Portion\ (Term, Answer) = 0$; thus equation (1) is verified.

**Period 2:** When any *checkpointing* request is received, any base station $MSS_i$ has to determinate the set *TempExtSeti* and immediately send the received portion *Term*, either to the initiator base station with the answer, or to the other base

stations with *checkpointing* requests. In this scenario, two cases can occur:

**Case 1**: if *TempExtSeti* = ∅, then the current base station sends an answer to the initiator base station with the value of: *Portion (Term, CkptReq)*.

**Case 2:** if *TempExtSet* ≠ ∅, then the base station broadcasts a *checkpointing* request to all other base stations (for each one) with a value of: *Portion (Term/m, CkptReq)*.
As the two cases cannot occur at the same time in the same base station,
$Portion\ (Term, MSS_n) = 0$;
$\sum Portion\ (Term, CkptReq) +$
$\sum Portion\ (Term, Answer) = 1$; hence, equation (1) is verified.

**Period 3:** As the number of mobile hosts in the system is limited, after completion time (as: $MH_i \in GlobSetj$, $\forall i \in [1, m]$ in the worst case), the number of *checkpointing* requests in transit in the network becomes nil. Therefore, after the reception of the last *checkpointing* request for all the base stations, we have:
$Portion\ (Term, MSS_n) = 0$;
$\sum Portion\ (Term, CkptReq) = 0$;
$\sum Portion\ (Term, Answer) = 1$; Thus equation (1) is again verified.

**Period 4:** As transmission times of the messages in the wired network are arbitrary but over, after completion time, all the answer requests arrive at the initiator base station. Hence,
$Portion\ (Term, MSS_n) = 1$;
$\sum Portion\ (Term, CkptReq) = 0$;
$\sum Portion\ (Term, Answer) = 0$; So, equation (1) is verified.

After completion (or, finished) time, from the initiation of the algorithm, the value of *Term* at the initiator base station becomes equal to 1. At this time, this initiator base station broadcasts the validation request. Upon receiving this request, every base station has to broadcast it in its own cellule. Then, every mobile host has to send its temporary checkpoint to its base station. Because of finished transmission message time, all the messages will arrive at their destinations. Consequently, a new global state is recorded, that is, the *checkpointing* algorithm ends after finished/completion time.

***Theorem*:** The algorithm insures recording of a consistent global state.

**Correctness proof:** To prove this theorem, we have to prove what follows:

If the reception event of a message was recorded in a checkpoint of a receiving mobile host, then the corresponding sending event of this message was also recorded in the checkpoint of the sender mobile host.

Let $MH_i \in Cell_i$ is covered by $MSS_i$, which is the mobile host receiving a message $m$, and let $MH_j \in Cell_j$ is covered by $MSS_j$, which is the sender of $m$. Then, $MH_j \in DepdSitesi$. If $MH_i$ records the reception of $m$ in $MH_i$'s checkpoint, then according to *Lemma 1*, the mobile host $MH_j$ has also recorded a checkpoint. Two possible cases are there for which the mobile host $MH_j$ establishes its checkpoint:

**Case 1**: Because of the reception of a *checkpointing* request coming from $MH_i$:

***Send*** *Message* ($MH_i$, $MH_j$, *Message)* by $MH_j \rightarrow$ ***Receive*** *Message* ($MH_i$, $MH_j$, *Message)* by $MH_i$.
***Receive*** *Message* ($MH_i$, $MH_j$, *Message)* by $MH_i \rightarrow$ ***Record*** *(TempCkpti)* by $MH_i$.
***Record*** *(TempCkpti)* by $MH_i \rightarrow$ ***Send*** *CkptReq (TempExtSeti, Value)* by $MSS_i$.
***Send*** *CkptReq (TempExtSeti, Value)* by $MSS_i \rightarrow$ ***Receive*** *CkptReq (TempExtSeti, Value)* by $MSS_j$.
***Receive*** *CkptReq (TempExtSeti, Value)* by $MSS_j \rightarrow$ ***Record*** *(TmpCkptj)* by $MH_j$.

Where '$\rightarrow$' represents the relation between operations (Lamport, 1978; Alagar and Venkatesan, 1997). As the relation $\rightarrow$ is transitive,

***Send*** *Message* ($MH_i$, $MH_j$,, *Message)* by $MH_j \rightarrow$ ***Record*** *(TempCkptj)* by $MH_j$.

Therefore, the sending of $m$ was saved in the checkpoint of the sender mobile host $MH_j$.

**Case 2:** Because of the reception of an application message $s$ from a mobile host $MH_k$, where $k \neq i$, then:

Let us suppose that $s$ was received and the local checkpoint was taken in $MH_j$ before $MH_j$ sends $m$ to $MH_i$. When $m$ arrives at $MSS_i$, the checkpoint number, piggybacked by $m$, is greater than that of the $MH_i$. So, the base station $MSS_i$ sent to $MH_i$ a *checkpointing* request before sending the message $m$; consequently, the mobile host $MH_i$ recorded its checkpoint before handling the message $m$. So, the reception of $m$ is not recorded in this checkpoint.

**Result**: If the reception event of a message was recorded in a checkpoint of a receiver mobile host, then the corresponding sending event was recorded in the *checkpointing* of the sender host. The absence of orphan message implies that the algorithm insures the recording of a consistent global state.

## 7. PERFORMANCE EVALUATION

### *7.1 Evaluation criteria*

To evaluate the performance of our algorithm, we used the following significant evaluation criteria:

**1. Technique of coordination used:** The *checkpointing* processes are not *blocking* in our algorithm as the inconsistency is resolved by the piggyback technique.

**2. Number of checkpoints created during the execution of the algorithm:** According to *Lemma 1*, the algorithm forces only those mobile hosts to record a checkpoint which are directly or indirectly dependent on the initiator mobile host. So, a minimum number (*Nmin*) of mobile hosts need to record checkpoints: $Nmin < n$.

**3. The cost of coordinating messages:** During the first phase, the mobile host which records a temporary checkpoint needs to send a *checkpointing* request to its base station and broadcast a *checkpointing* request in its cellule. Hence, the incurred cost of the coordinating messages transmitted in the system, during the first phase, is about: $Nmin \times (CstWd + CstBr)$.

- *CstWd*: cost of sending a coordinating message in the wired network.
- *CstBr*: cost of broadcasting a coordinating message in the wireless network to all mobile hosts.

During the second phase, the initiator base station sends a request of validation to all the base stations. Every base station, having received the validation request, has to broadcast, in turn, this request in its own cellule. Every mobile host in the cellule, having previously recorded a temporary checkpoint, has to send this temporary checkpoint to its base station. So, the incurred cost of the coordinating messages transmitted in the system, during this second phase, is equal to: $m \times (CstWd + CstBr) + Nmin \times CstWs$.

- *CstWs*: cost of sending a coordinating message in the wireless network.

Thus, the total cost of coordinating messages is about: $(Nmin + m) \times (CstWd + CstBr) + Ndep \times CstWs$.

**4. Time of blocking:** The blocking time of our algorithm is nil.

**5. Distribution of the algorithm:** According to the way of operation, the algorithm is fully distributed.

**6. Minimum Process:** Our algorithm requires a minimum number of mobile hosts to participate in *checkpointing*, and also takes into account the limited storage available at MH.

**7. Useless Checkpoint:** No local checkpoint is useless after the execution of our algorithm. A useless checkpoint is a local checkpoint that cannot be part of a consistent global state.

**8. Piggyback Information:** The algorithm piggybacks only an integer checkpoint sequence number (*Ckpti*) with each application message.

**9. Concurrent Executions:** Considering the low probability of multiple and concurrent failures, our algorithm can handle at most one execution at the same time.

*7.2 Comparative study with related work*

A comparison, in terms of performance, of our *checkpointing* algorithm with regard to the other algorithms with the same class is presented in this section. Thus, the Table 1 compares the performance of our algorithm with eight *checkpointing* algorithms fitting the class of coordinated *checkpointing*, using the same previous parameters.

We notice that Koo and Toueg algorithm (Koo and Toueg, 1987), requires only minimum number of MHs ($\leq Nmin$) to record a checkpoint, so that the cost of coordinating messages is reduced, but the execution of this algorithm needs to block the execution of the distributed application for a long time ($Nmin \times Tck$), which reduces the performance of the application. Also, the coordinated *checkpointing* algorithm proposed in (Kiehn et al., 2014) requires only a minimum number of MHs to take a checkpoint. This scheme designed by the generalization of Chandy and Lamport's algorithm (Chandy and Lamport, 1985), uses a delayed *checkpointing* approach, along with partial channel blocking to achieve the consistent checkpoints. But, this blockage can be impossible for some type of distributed application.

The Elnozahy et al. algorithm (Elnozahy et al., 1992) and Neogy (Neogy et al., 2004) algorithm, which are *non-blocking*, force all MHs in the system ($n$) to take checkpoints; our algorithm reduces this number to nearly minimum ($Nmin \leq n$), so that the total number of checkpoints transmitted into the system is reduced. However, for a mobile environment, it is also very critical to minimize the number of coordinating messages transmitted. Hence, if we only consider the cost of coordinating messages sent to the mobile link, our algorithm performs fairly.

The *non-blocking* algorithms in (Neves and Fuchs, 1997) and in (Surender et al., 2013), involve a minimum number of MHs (greater than the minimum involved in the Koo and Toueg algorithm (Koo and Toueg, 1987)) to record a checkpoint. At the same time, the cost of coordinating messages of these algorithms is the smallest compared to the other algorithms. This is because, these algorithms use time to indirectly coordinate to minimize the number of coordinating messages transmitted through the system. But, the timers in the MH cannot be perfectly synchronized – in fact, it is still an open research issue; therefore, the consistency between all the checkpoints can still be a hard problem for this type of algorithms.

Biswas and Neogy design in (Biswas and Neogy, 2013) an algorithm based on the message logging and independent *checkpointing*. The authors also present a cryptographic method for securing checkpoints and logs. In this algorithm, MSS logs all the messages sent to each MH in the stable storage. Additionally, each MH also takes checkpoints to reduce the extent of rollback during recovery. If a failure occurs, only the failed MH recovers by using the checkpoints and logged messages to replay the events precisely as they occurred during the pre-failure execution period. But, during this re-execution, the MH resends any message that it sent during this same execution stage before the failure. This actually incurs a real disadvantage for a mobile network, which has low bandwidth and limited energy resource. The second disadvantage of this algorithm is that when the mobility rate of MHs is high, the message logs of an MH may be distributed on several MSSs. Hence, after a failure occurs, the current MSS needs to collect all the message logs of MH. This may increase the cost of such operations which may be unfeasible especially for highly distributed applications. Li et al. propose in (Li et al., 2014) a *checkpointing* message logging scheme considering the *visit time* of the MH to the MSS. This scheme also suffers from the same disadvantages that were mentioned earlier. In fact, this scheme works efficiently only if the stay times of the MHs in the MSSs are long enough - in other words, if the mobility rate of the MHs is less.

**Table 1. Performance comparaison.**

| Algorithm | Technique Used | Number of Checkpoints | Message Cost | Blocking Time | Distributed | Minimum Process | Useless Checkpoint | Piggyback Information | Concurrent Executions |
|---|---|---|---|---|---|---|---|---|---|
| (Koo and Toueg, 1987) | *Blocking* | $\leq Nmin$ | $3 \times Nmin \times Ndep \times (CstWd + CstWs)$ | $(Nmin \times Tck)$ | *Yes* | *Yes* | *No* | *Integer* | *No* |
| (Elnozahy et al., 1992) | *Non blocking* | $n$ | $2 \times CstBr + n \times (CstWd + CstWs)$ | 0 | *No* | *No* | *No* | *Integer* | *No* |
| (Neves and Fuchs, 1997) | *Time based* | $n$ | $2 \times Nmin \times (CstWd + CstWs)$ | 0 | *Yes* | *No* | *Present* | *Integer* | *No* |
| (Cao and Singhal, 2001) | *Non blocking* | $Nmin$ | $\approx 2 \times Nmin \times CstWd + min\,(Nmin \times (CstWd + CstWs), CstBr)$ | 0 | *Yes* | *Yes* | *Present* | *Integer* | *Yes* |
| (Neogy et al., 2004) | *Non blocking* | $n \times (CstWs + CstWd + CstBr)$ | $2 \times CstBr + n \times (CstWs + CstWd + CstBr)$ | 0 | *Yes* | *No* | *No* | *Integer* | *No* |
| (Kumar et al., 2010) | *Non blocking* | $Nmin$ | $3 \times Nmin \times (CstWs + CstWd + CstBr)$ | 0 | *Yes* | *Yes* | *No* | *Integer* | *No* |
| (Nagpal et al., 2013) | *Blocking* | $\leq Nmin$ | $2 \times CstWs + 3 \times CstBr + 4 \times Nmin \times CstWs + 2 \times m \times CstWd$ | $\leq (Nmin \times Tck)$ | *Yes* | *Yes* | *No* | *Integer* | *No* |
| (Surender et al., 2013) | *Time based* | $\approx Nmin$ | $Nmin \times (CstWd + CstWs)$ | 0 | *Yes* | *Yes* | *Present* | *Integer* | *No* |
| **Our Algorithm** | ***Non blocking*** | ***Nmin*** | $\mathbf{(Nmin + m) \times (CstWd + CstBr) + Ndep \times CstWs}$ | **0** | ***Yes*** | ***Yes*** | ***No*** | ***Integer*** | ***No*** |

An effort is made in the three phase minimum-process coordinated *checkpointing* algorithm proposed in (Nagpal et al., 2013) to minimize the blocking time of the distributed application and the useless checkpoints compared to the other *non-blocking* scheme proposed in (Koo and Toueg, 1987). However, this is at the expense of coordinating message cost (i.e., relatively higher cost compared to the other algorithms), which may cause exhaustion of the mobile network.

Compared to Cao and Singhal algorithm (Cao and Singhal, 2001) and Kumar et al. algorithm (Kumar et al., 2010), which are also *non-blocking* and offer the minimum, our algorithm requires almost the same number of MHs to record a checkpoint. However, it requires less number of coordinating messages. Therefore, the cost to ensure a consistent global state is less, in other words:

$$(Nmin + m) \times (CstWd + CstBr) + Ndep \times CstWs < 3 \times Nmin \times (CstWs + CstWd + CstBr) \quad (2)$$

and:

$$(Nmin + m) \times (CstWd + CstBr) + Ndep \times CstWs < 2 \times Nmin \times CstWd + min\,(Nmin \times (CstWd + CstWs), CstBr) \quad (3)$$

From (Kumar et al., 2010), the proposed algorithm in (Kumar et al., 2010) generates the consistent global state with approximately the same message cost as in the algorithm proposed by (Cao and Singhal, 2001):

$$3 \times Nmin \times (CstWd + CstBr + CstWs) \approx 2 \times Nmin \times CstWd + min\,(Nmin \times (CstWd + CstWs), CstBr) \quad (4)$$

So, from equation (4), the certainty verification of equation (2) is sufficient to ensure the certainty of equation (3). It is known that for any minimum *checkpointing* algorithm, we have:

$$\begin{aligned} &Ndep \leq Nmin \Rightarrow Ndep \times CstWs \leq Nmin \times CstWs \\ &\Rightarrow (CstWd + CstBr) + Ndep \times CstWs \\ &\leq (CstWd + CstBr) + Nmin \times CstWs \\ &\Rightarrow Nmin \times (CstWd + CstBr) + Ndep \times CstWs \\ &\leq Nmin \times (CstWd + CstBr) + Nmin \times CstWs \\ &\Rightarrow Nmin \times (CstWd + CstBr) + Ndep \times CstWs \\ &\leq Nmin \times (CstWd + CstBr + CstWs) \\ &\Rightarrow Nmin \times (CstWd + CstBr) + Ndep \times CstWs < \\ &3 \times Nmin \times (CstWd + CstBr + CstWs) \end{aligned} \quad (5)$$

As, mathematically, equation (5) is always verified, it remains to verify equation (2) in relation to the number of base stations in the system, $m$ through simulation experiments, presented in the following section.

It is worth mentioning here that some other works, for instance, Sharma et al. (2014) show that inconsistency exists in Cao and Singhal's algorithm (Cao and Singhal, 2001).

*7.3 Simulation setting and results*

This section presents the performance evaluation of our algorithm performed through the simulation experiments. A series of experiments were done to find out the efficiency of our scheme. The simulation environment comprises a variant number of MSSs and MHs. We assume that there is one process running on each MH. The MSSs are connected by a wired network with a bandwidth of 10 mbps. Each MH has a wireless connection with its supporting MSS with a bandwidth of 2 mbps, which follows IEEE 802.11 standard. The size of each computation message is assumed to be 2 kb. During *checkpointing*, there are coordination messages transmitted among the processes for checkpoint requests and responses. Each coordination message is assumed to be 100 bytes. Therefore, the transmission delay of messages is: 4 ms and the size of a checkpoint is: 512 kb.

The information of a particular MH gets scattered over a number of MSSs. So, the message cost depends on the number of MHs and MSSs in the system from which information is to be collected (Biswas and Neogy, 2011). Thus, in our experiments, we changed the number of MSSs (10, 30, and 60) and varied the number of MHs (from 10 to $10 \times m$) to see the corresponding changes of the message cost and to compare the performance of our algorithm with Cao and Singhal algorithm (Cao and Singhal, 2001) and Kumar et al. algorithm (Kumar et al., 2010).

Fig. 4 presents the cost of coordinating messages to complete the *checkpointing* processes in the best case scenario for the three algorithms against the number of MHs in the system, with varying number of MSSs.

From the Fig. 4, it can be noticed that the three algorithms require a higher message cost for determining a consistent state as the numbers of MSSs and/or MHs in the system increase. We can also see that the message costs of the proposed algorithm in (Cao and Singhal, 2001) and in (Kumar et al., 2010) are almost identical. The advantage of our algorithm over the other schemes is that the message cost is relatively much lesser as the numbers of MSSs and/or MHs decrease. In other words, when the system contains a less number of MSSs and/or MHs, our algorithm is relatively more suitable because it requires low message cost to save a consistent global state.

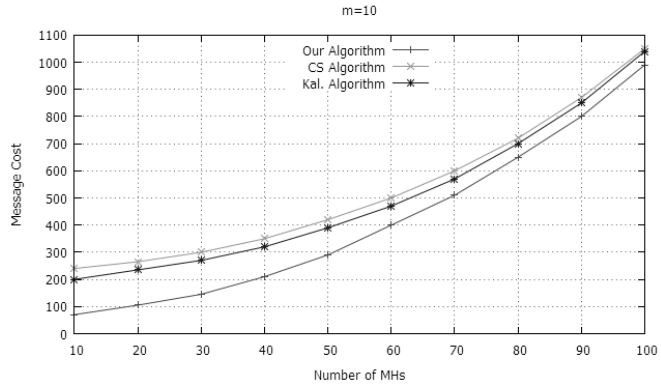
**A. Advantages of our algorithm:**

- More than the numbers of MSSs and/or MHs in the system are less, our algorithm reduces the cost of coordinating messages until: $(Nmin + m) \times (CstWd + CstBr) + Ndep \times CstWs$.
- Our algorithm reduces the number of checkpoints until *Nmin*, knowing that: $Ndep \leq Nmin \leq n$.
- The time of blocking is nil. Our algorithm complies in a transparent way in the distributed applications, which indeed increases performance in a significant way.

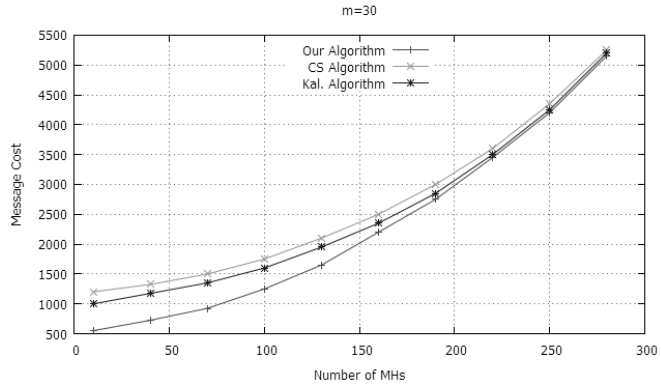
**B. Shortcomings of our algorithm:**

- Our algorithm cannot handle concurrent failures and multiple executions at the same time. This is an issue that we have kept for our future works.
- The cost of coordinating messages in our algorithm increases and reaches the cost of other schemes whenever the

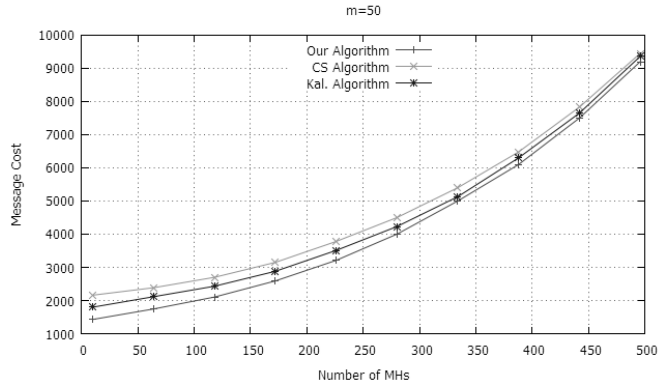
numbers of MSSs and/or MHs in the system increase. Still, for most of the application scenarios, our scheme performs relatively better.




(a)




(b)




(c)

Fig. 4. Message cost vs. number of MH in the system when: (a) $m = 10$, (b) $m = 30$, (c) $m = 50$.

## 8. CONCLUSIONS AND FUTURE WORKS

In a mobile computing environment, mobile hosts may be exposed to problems coming from limited and unstable storage supports, a weak bandwidth, a limitation of energy source, high rate of errors, frequent movements, and disconnections. A *checkpointing* algorithm suitably adapted to this context has to take into account these different constraints. Hence, the unavailability of a mobile host (because of disconnection) can establish a sufficient criterion to be able to invalidate the adoption of a *blocking checkpointing* algorithm, notably when application is time-bounded. Also, to save some energy, certain mobile hosts involved in the computation can be in doze mode. Any coordinating message directed to them may lead to useless activation of them. Therefore, a *checkpointing* algorithm that requires all the hosts to take checkpoints systematically would keep some of them awake unnecessarily. This was the core motivation of our proposal of a *checkpointing* algorithm which brings a solution to the problems previously raised in this research arena.

Our proposed algorithm belongs to the coordinated type, minimizes the number of recorded checkpoints, and avoids blocking the mobile hosts. It allows transferring all the payload of coordination towards the base stations of the wired network, which significantly decreases the amount of traffic on the mobile network for the benefit of other necessary activities. The algorithm also uses the information of dependence to limit, to the strict necessities, the number of mobile hosts to take checkpoints. Consequently, incurred overhead due to the activities relating to the *checkpointing* process is largely reduced. As future works, we would like to overcome the shortcomings of our scheme – especially, on handling concurrent failures and multiple executions at the same time, and further reducing the cost of coordinating messages.

## ACKNOWLEDGMENT

This work was partially supported by Networking and Distributed Computing Laboratory (NDC Lab.), KICT, IIUM.

## REFERENCES

Acharya, A. and Badrinath, B.R. (1994). Checkpointing distributed applications on mobile computers. In: Proc. *3rd International Conference on Parallel and Distributed Information Systems,* Texas, USA, 73–80.

Alagar, S. and Venkatesan, S. (1997). Causal ordering in distributed mobile systems. *IEEE Transactions on Computers, Special issue on Mobile Computing,* 46 (3), 353–361.

Awasthi, L.K., Misra, M. and Joshi, R.C. (2014). Minimum mutable checkpoint-based coordinated checkpointing protocol for mobile distributed systems. *International Journal of Communication Networks and Distributed Systems,* 12 (4), 356–380.

Badrinath, B.R., Acharya, A. and Imieliński, T. (1993). Impact of mobility on distributed computations. *ACM SIGOPS Operating Systems Review*, 27 (2), 15–20.

Badrinath, B.R., Acharya, A. and Imieliński, T. (1994). Structuring distributed algorithms for mobile hosts. In: Proc: *14th International Conference on Distributed Computing Systems,* Pozman, Poland, 21–28.

Biswas, S. and Neogy, S. (2011). A handoff based checkpointing and failure recovery scheme in mobile computing system. In: Proc: *2011 IEEE International*

*Conference on Information Networking,* Barcelona, Spain, 441–446.

Biswas, S. and Neogy, S. (2013). Improving recovery probability of mobile hosts using secure checkpointing. In: Proc: *2nd International Conference on Advances in Computing, Communication and Informatics,* Mysore, India, 984–989.

Cao, G. and Singhal, M. (2001). Mutable checkpoints: A new checkpointing approach for mobile computing systems. *IEEE Transactions on Parallel and Distributed Systems,* 12 (2), 157-172.

Chandy, K.M. and Lamport, L. (1985). Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems,* 3 (1), 63–75.

Dey, P. and Biswas, S. (2012). Handoff based secure checkpointing and log based rollback recovery for mobile hosts. *International Journal of Network Security & Its Applications,* 4 (5), 25–41.

Elnozahy, E.N., Alvisi, L., Wang, Y.M. and Johnson, D.B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys,* 34 (3), 375–408.

Elnozahy, E.N., Johnson, D.B. and Zwaenepoel, W. (1992). The performance of consistent checkpointing. In: Proc. *11th Symposium on Reliable Distributed Systems,* Texas, USA, 39–47.

Forman, G.H. and Zahorjan, J. (1994). The challenges of mobile computing. *IEEE Computer,* 27 (4), 38–47.

Huang, S.T. (1989). Detecting termination of distributed computations by external agents. In: Proc. *9th International Conference on Distributed Computing Systems,* California, USA, 79–84.

Kanmani, P., Anitha, R. and Ganesan, R. (2007). Coordinated checkpointing with avalanche avoidance for distributed mobile computing system. In: Proc. *International Conference on Computational Intelligence and Multimedia Applications,* Tamilnadu, India, 461–463.

Kiehn, A., Raj, P. and Singh, P. (2014). A causal checkpointing algorithm for mobile computing environments. In: Proc. *15th International Conference on Distributed Computing and Networking,* Coimbatore, India, 134–148.

Koo, R. and Toueg, S. (1987). Checkpointing and rollback-recovery for distributed systems. *IEEE Transactions on Software Engineering,* 13 (1), 23–31.

Kumar, S., Chauhan, R.K. and Kumar, P. (2010). A low overhead minimum process global snapshot collection algorithm for mobile distributed systems. *The International Journal of Multimedia & Its Applications,* 2 (2), 12–30.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communication of the ACM,* 21 (7), 558–565.

Li, X., Yang, M., Men, C.G., Jiang, Y.T. and Udagepola, K. (2014). Access-Pattern aware checkpointing data storage scheme for mobile computing environment. In: Proc. *11th International Conference on Mobile Systems and Pervasive Computing,* Niagara Falls, Canada, 330–337.

Nagpal, M., Kumar, P. and Jangra, S. (2013). Three phases coordinated checkpointing scheme for mobile distributed systems. *International Journal of Latest Transactions in Engineering and Science,* 1 (2), 1–8.

Neogy, S., Sinha, A. and Das, P.K. (2004). A checkpointing protocol for distributed system processes. In: Proc. *IEEE Region 10 Conference,* B (2), Bangkok, Thailand, 553–556.

Netzer, R.H.B. and Xu, J. (1995). Necessary and sufficient conditions for consistent global snapshots. *IEEE Transactions on Parallel and Distributed Systems,* 6 (2), 165–169.

Neves, N. and Fuchs, W.K. (1997). Adaptive recovery for mobile environments. *Communication of the ACM,* 40 (1), 68–74.

Park, T., Woo, N. and Yeom, H.Y. (2002). An efficient optimistic message logging scheme for recoverable mobile computing systems. *IEEE Transactions on mobile computing,* 1 (4), 265–277.

Schlichting, R.D. and Schneider, F.B. (1983). Fail-stop processors: An approach to designing fault tolerant distributed computing systems. *ACM Transactions on Computer Systems,* 1 (3), 222–238.

Sharma, P.K., Kumar, P. and Jangra, S. (2014). Proxy MSS based synchronous checkpointing approach for mobile distributed systems. *International Journal of Innovations in Engineering and Technology,* 3 (4), 194–202.

Surender, J., Arvind, S., Anil, K. and Yashwant, S. (2013). Low overhead time coordinated checkpointing algorithm for mobile distributed systems. In Chaki, N., Meghanathan, N. and Nagamalai, D. (ed.), *Computer Networks & Communications (NetCom),* 173–182. Springer, New York.