Consistent Global States of Distributed Mobile Computations*

Zhonghua Yang, Chengzheng Sun, Abdul Sattar, and Yanyan Yang

School of Computing & Information Technology
Griffith University, Brisbane,
Qld 4111 Australia
Email: {z.yang, c.sun, a.sattar, yyang}@cit.gu.edu.au

Abstract

In this paper, we consider one of fundamental issues in distributed systems, that is, calculating consistent global states, or consistent distributed snapshots. Two algorithms of distributed snapshots are presented using a two-phase-cut approach. These two algorithms adopt a two-tier system model, separating the system into two parts: a resource-rich network consisting of stationary hosts and the wireless networks consisting of resource-poor mobile hosts and supporting stationary hosts. The algorithms rely on the resource-rich network to take an active role and substantially undertake the responsibility for distributed snapshots and for calculating consistent global states, The two-phase-cut algorithms employ an efficient message passing mechanisms with low overhead. How the specific mobile issues identified in the paper are handled is also discussed.

1. Introduction

A distributed system consists of several processes that execute on geographically dispersed computers and collaborate via message-passing with each other to achieve a common goal. In a traditional distributed system all hosts are stationary. Recent advances in portable computers with wireless communication interfaces and satellite services have made it possible for mobile users (mobile computers) to perform distributed applications and to access information anywhere and at anytime. This new computing environment where some hosts are mobile computers connected by wireless communication networks and some are stationary computers connected by a fixed network is called a *distributed mobile computing environment*. Thus, a distributed mobile system can be considered as a special kind of general distributed systems where some of its hosts are not fixed in their location. This new paradigm is *distributed mobile computing*. Clearly, a mobile system is not necessarily a distributed system, and mobile computing is not necessarily distributed computing.

A distributed mobile system is characterized by the *mobility* and *poor resource* of mobile hosts. These two distinct features raise various new issues and constraints not faced in a stationary distributed system [16,15]. In this paper we identify the following four issues and discuss their implications when designing algorithms in distributed mobile setting.

Issue 1. Mobile connectivity is highly variable in performance and reliability (e.g. communication delay), and the wireless communication channels used by the mobile hosts have a lower bandwidth than the wire-line, fixed communication links between stationary systems.

Thus, the burden of computation and communication load cannot be distributed equally among stationary and mobile hosts.

Issue 2. Mobility is inherently vulnerable. The disk storage is potentially unstable for logging or recording of the states. For example, a laptop is accidentally physically dropped or stolen; or the data stored on a mobile host's disk are totally wiped out by some security systems.

Thus, the state recording and message logging cannot rely on the mobile host's storage, and the saved local states and message logs cannot be

* The early version of this paper appeared in *the Proceedings of The 1998 International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA'98).* Las Vegas, Nevada, USA. July 13-16, 1998.

assumed to be immediately available from the mobile hosts when it is required.

Issue 3. Mobile hosts are often disconnected from the rest of the system. A disconnected mobile host can neither send nor receive messages, but can continue an application execution by using its local data and cashed shared data [9]. Disconnected operations become a regular feature in mobile computing. Disconnected operations in a mobile environment is distinct from failure in that they are voluntary in nature and a mobile host can be required to execute a *disconnection protocol* before its detachment.

Thus, The algorithms have to accommodate this voluntary disconnection and make progress during the disconnection of mobile hosts.

Issue 4. The mobility implies that a mobile host may change its location during distributed computation.

Thus, the location management or search for the targeted mobile host becomes an indispensable task of any distributed protocols/algorithms.

All these new issues and challenges have made those algorithms devised for traditional distributed systems not applicable. In this paper, we consider one of fundamental issues in distributed computations: calculating the *consistent global states*, which underlying many distributed applications. Calculating global states is sometimes called taking *distributed snapshots* (a distributed snapshot returns a global state). A large class of important problems in distributed systems can be cast as periodically calculating consistent global states and executing some reactions based on the global state that have been taken. Examples of such problems include distributed debugging and monitoring, fault-tolerant and rollback-based recovery, detection of state properties such as a deadlock and termination. This paradigm requires consistently recording (often, periodically recording) the global state of a distributed computing. A global state is a collection (union) of the local states, one from each process of the computation, recorded by a process. The global state is said to be *consistent* if it looks to all the processes as if it were taken *at the same instant everywhere in the system*. There have been many papers on finding consistent global states of a distributed application [17]. However, the constraints imposed by the mobility and poor-resource of mobile hosts as outlined above complicate the design of distributed algorithms and applications, and make them inappropriate for distributed mobile computing environments. In this paper, we present two distributed algorithms for constructing a consistent global state of distributed mobile applications. Our algorithms tackle *the issue 1* by adopting a two-tier system model: a stationary-tier and a mobile-tier. In this model, the stationary tier (hosts) is required to play a key and guiding role by undertaking the major communication and computation of the algorithms and store the local states of all the mobile hosts. On the other hand, the mobile-tier (hosts) plays only the passive role and undertakes the communication and computation only when necessary. The local states recorded by the mobile hosts are immediately sent to the supporting stationary hosts and is not kept in the mobile hosts, as a solution to *the issue 2*. Our algorithms allow the voluntary disconnected operations simply by requiring the mobile host to take a local snapshot and send to its supporting stationary host *before* disconnection (*the issue 3*). Finally, our algorithms incorporate a simple *handoff* protocol to handle the changing of mobile host's location as a solution to the location management issue (the *issue 4*). As analyzed in the paper, our algorithms are designed to be very message efficient, adding a light burden on the mobile host.

The rest of this paper is organized as follows. In the next section we present a two-tier mobile system model based on a configuration of the system having a fixed network part and a wireless network part. In Section 3, two Two-Phase-Cut algorithms for distributed snapshots are presented. The algorithms are analyzed and shown that they are *message efficient*. The approach to tackling the mobility issues is discussed in Section 4 and 5. Some related work is remarked in Section 6. Finally we conclude the paper in Section 7.

2. System Model

The system model that we adopt in this paper is adapted from the architectures used in [4,8].

A distributed mobile system is described as a two-tier model consisting of a set *of mobile hosts (MHs)* and *stationary hosts*. A mobile host is a host which is *able to move* (can change its location with time) *while retaining its network connections* [8]. A stationary host, as its name implies, does not change its location and connects/communicates to other stationary hosts via a wired *fixed network*. We do not require that the fixed network is completely connected, but the underlying protocols allow a stationary host to communicate with any other stationary host. The communication channels are not required to be FIFO. Some stationary hosts also serve as an infrastructure computer to support communication between mobile hosts and is called a *mobile support system (MSS).* A MSS communicates directly with the MHs through a wireless channel. A

MSS and all the MHs it supports form a *cell* or *wireless cell*, which is generally a logical or geographical area covered by a MSS. The wireless medium allows a MSS to communicate with all the MHs within its cell with a single message transmission. For simplicity of presentation, we assume that all nodes in the fixed network are the MSS. All MHs that have identified themselves with a MSS belong to its cell and are considered *local* to the MSS. A MH may belong to only one cell at any given time. A MH can communicate with other MHs and MSSs only through the MSS to which it belongs. The communication between a MSS and an MH needs not be FIFO, either. All communications in the fixed network are assumed reliable; the communication primitives of multicast or broadcast are not available. Communications within wireless cell tend to be error prone and not very reliable, but a simple protocol could be devised to make wireless communication reliable. In the presentation below, therefore, we focus on the consistent snapshot algorithms, and will also make an assumption of *reliable* communication within wireless cell. Message transmission delays are unpredictable but finite. A two-tier distributed mobile system is shown in Figure 1.

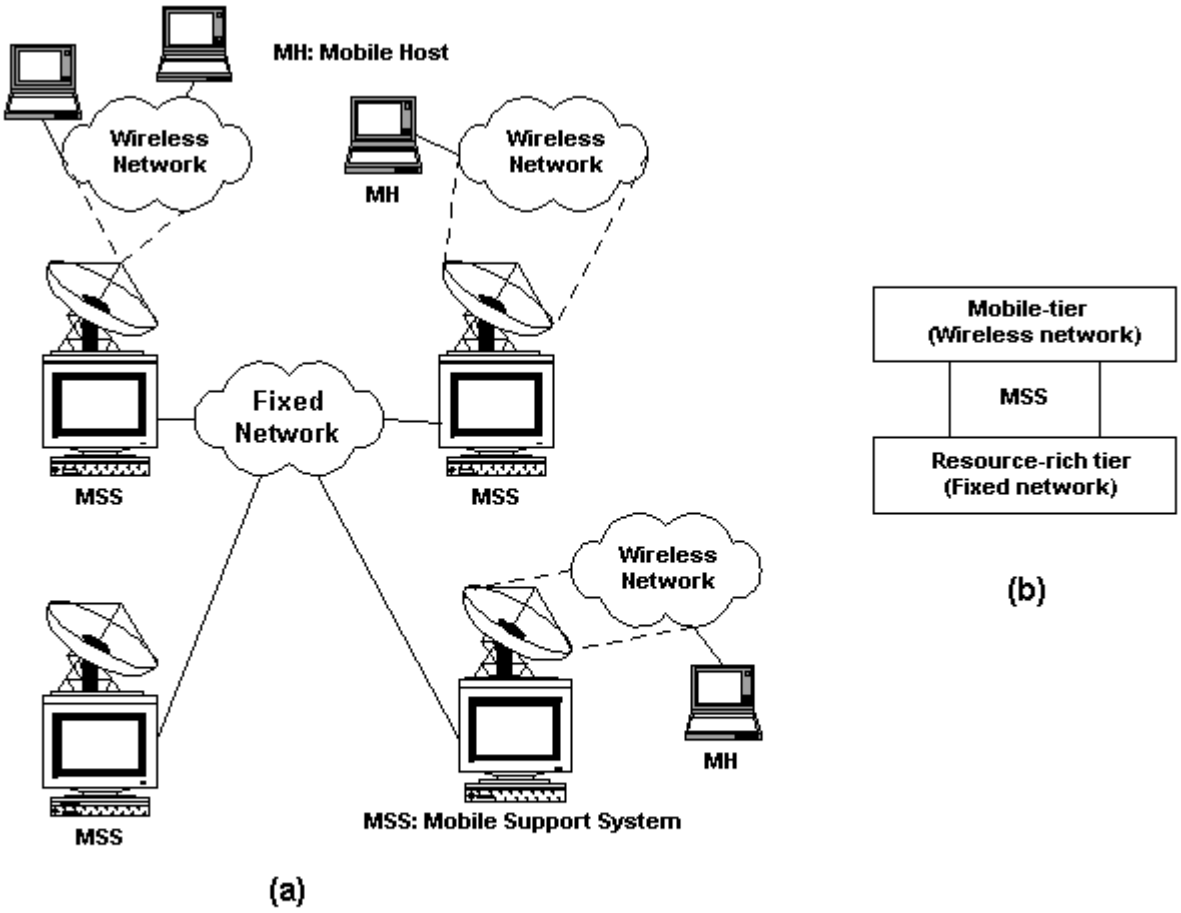

Figure 1. (a) The configuration of a distributed mobile system
(b) A two-tier model for the distributed mobile system

In this system model, we make an architectural distinction between the resource-rich fixed network and the resource-poor tier. As described in the next section, our algorithms take this architectural advantage to tackle the mobile issues by relying on the resource-rich network to take an active role and to substantially undertake the responsibility for distributed snapshots and for calculating consistent global states. On the other hand, the mobile-tier (hosts) plays only the passive role and undertakes the communication and computation *only when* necessary.

A distributed application consists of a set of processes $\{p_1, p_2, \ldots, p_n\}$ that run on different hosts of the distributed mobile system. In the following, the term *host*, *node* and *process* will be used interchangeably. The processes communicate with each other via message-passing. The system does not have a global clock and global time is not available to applications.

Each process in the system executes a sequence of *events* that brings the process from one state to another. There are three kinds of *events* of interest in a system, *send*, *receive*, and *internal events*. We denote xth event in process p_i by e_i^x. The occurrences of these events are governed by the *happen-before* relation [10], denoted by $\rightarrow$, which is defined below.

Definition 1 *Event e_i^x happen-before e_j^y, denoted by $e_i^x \rightarrow e_j^y$, if (1) $i = j$ and $x < y$, or (2) e_i^x is the sending of a message and e_j^y is the corresponding receiving of that message, or (3) there is an event e_i^z such that $e_i^x \rightarrow e_i^z$ and $e_i^z \rightarrow e_j^y$.*

The causality between events as defined by *happen-before* relation can be used to describe mutually consistent local states of a distributed system. A set of local states is said to be mutually *consistent* if it looks to all the processes as if it were taken *at the same instant everywhere in the system*. However, since the global real time is not available in distributed systems, we cannot rely on setting a common *time instant* to determine the consistency of local states. Instead, we use the causality between events to obtain the desired consistency. But first we give a formal definition of a global state and its consistency.

For each process p_i, the *local state*, LS_i of p_i at a given time is defined by the local context of the distributed application. The global state of a distributed mobile computing is a collection of all the local states. More formally,

Definition 2 *A global state of a distributed mobile computation is a set of local state, $GS = \{LS_1, LS_2, \ldots, LS_n\}$, each per process in the computation.*

Note that in a distributed system only local states of processes are (locally) *observable* at a given time. Therefore, the consistency has to be considered with respect to the local states of processes. For two communicating processes, p_i and p_j, and p_i has sent a message m_{ij} to p_j, their local states (LS_i and LS_j) are *inconsistent* if and only if $m_{ij} \notin LS_i$ and $m_{ij} \in LS_j$. We denote by *inconsistent* (LS_i, LS_j) the set of message-receive events in LS_j without the corresponding send-events in LS_i. In other words, if a message-receiving event is included in a local state LS_j of the receiving process but the corresponding send-event is not included in a local state LS_i of the sending process, then LS_i and LS_j are clearly not consistent. Thus, we have:

Definition 3 *A global state $GS = \{LS_1, LS_2, \ldots, LS_n\}$, is consistent if, and only if, $\forall i \ \forall j$: inconsistent$(LS_i, LS_j) = \varnothing$.*

On the other hand, it is possible that a message is recorded in a local state of the sending process but has not been included in the local state of the receiving process. In this case, the message is held in the channel between the two communicating processes, and the message is a *transit* message. The set of messages in transit between two processes is denoted by *transit* (LS_i, LS_j).

Definition 4 *A global state $GS = \{LS_i, LS_2, \ldots, LS_n\}$ is transitless if, and only if, $\forall i \ \forall j$: transit$(LS_i, LS_j) = \varnothing$.*

Definition 5 *A global state is strongly consistent if, and only if, it is consistent and transitless [6].*

This last property means that, for any pair (LS_i, LS_j), a message m_{ij} is received in LS_j if, and only if, it is sent in LS_i.

In the following, we assume that the local state recorded by a process includes the channel state (the messages held in the channel) and will not separately consider the recording of the channel states.

In addition, we do not consider issues related to process failure. Like the most (if not all) of distributed algorithms that calculate the consistent global states, our algorithms, as described in the next section, are not designed to accommodate process failures that occur during the algorithm execution.

3. Two Phase Cut Algorithms

In this section, we present two algorithms for finding consistent global states of a distributed mobile computation, called *two phase cut algorithms*. As indicated earlier, the algorithms deal with mobility issues mentioned above and rely on the fixed MSS systems to take responsibility for initiating the snapshot, for guiding the progress of the snapshot process, and for constructing the consistent global states from collected local states. As a result of adopting the two-tier system model, the communication and computation of mobile hosts is minimal.

In both algorithms, three sets of messages, *Prepare*, *Cut*, and *Resume*, are sent respectively during the two phases: from the initiator to all processes, back to the initiator, and back to processes. The sending of application messages is disabled during taking snapshot. These messages take the form of *(Prepare, {snp_{no}, MSS_{id} }), (Cut, {snp_{no}, MSS_{id}}),* and *(Resume, {snp_{no}, MSS_{id}}),* where the pair *{snp_{no}, MSS_{id}}* denotes the snapshot number initiated by MSS whose process id is MSS_{id}.

3.1. Prepare-and-Cut Algorithm

The first Two-Phase-Cut algorithm is a Prepare-and-cut algorithm, so named because of two distinct phase messages *Prepare* and *Cut*. The messages from the initiator are indirectly sent to the other MSS along paths of length *d* or less, where *d* is the diameter of the fixed network. Recall that we do not assume that the network is completely connected, and we also do not assume that the broadcast is available.

We assume that the snapshot is initiated by one of MSSs. Typically, the initiator is a monitor or coordinator process in the system. Its designation is application dependent. In this paper, we simply assume that there exists an initiator that initiates a distributed snapshot.

The following procedures prescribe the actions taken by the initiator, the MSS, and MHs respectively

Actions taken by the Initiator:

1. The initiator executes a *StartCut* event (to signify the beginning of a snapshot) and then sends *Prepare* to each other stationary process (MSS); It also sends *Prepare* to all its own MHs.

2. Waits for *Cut* messages from all processes including its MHs.

3. After receiving *Cut* from all processes, immediately records its state, and then sends *Resume* to all other MSS processes.

Actions taken by the responding MSS:

1. Upon receiving *Prepare*, disables the sending of application messages (including the forwarding of application messages to its MHs), and records its state.

2. Send *Prepare* to all MHs within its cell, and wait for *Cut* from all its MHs;

3. After receiving *Cut* from all its MHs, sends *Cut* to the initiator, and then wait for *Resume* from the initiator.

4. Upon receiving *Resume*, re-enables the sending of application messages and sends *Resume* to its MHs who will re-enables the sending of application messages.

Actions taken by the mobile host:

1. Upon receiving *Prepare* from its MSS, disables the sending of application messages and records its state,

2. Sends *Cut* to its MSS, and then wait for *Resume* message from its MSS.

3. Upon receiving *Resume*, re-enables the sending of application messages.

4. Sends the local state to its MSS.

After each host has taken snapshots, the initiator can collect them to form a global state that is consistent by the algorithm.

Theorem 1 *The global state collected by the initiator after the Prepare-and-Cut Algorithm completes is a consistent global state.*

Proof. *The key here for a global state to be consistent is to show that if an event of the message-receive by a process is included in a global state, then its corresponding sending event is also included in the global state. To see this is the case by our algorithm, we prove it by contradiction. For the simplicity of presentation, we assume that the sending process p_i is the Initiator. Suppose that m_1 sent by p_i is received by*

process p_j and included in p_j's snapshot and it was received by p_j before it received Prepare sent by p_i (Figure 2). If the global state by the algorithm is inconsistent, then the sending of m_1 must be after p_i sending out the Resume. But this could not happen because during the period of Prepare to Resume, the processes including p_i are disabled to send any application message: a process could not receive a message (e.g. m_1) which was not being sent, thus contradicting the assumption. ■

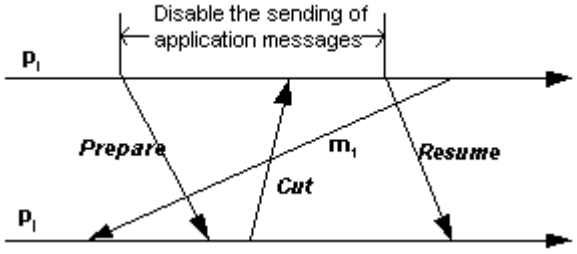

Figure 2. Proof by contradiction: An impossible scenario by the algorithm where a receive is in the snapshot, but the corresponding send is not.

3.2 Cut-Along-Tree Algorithm

The second algorithm for calculating consistent global states is called the *Cut-Along-Tree* algorithm. As the name implied, in the algorithm, the *Prepare*, *Cut*, and *Resume* messages are sent along a spanning tree of the fixed network; consequently, it requires bi-directional channels so that messages may take the same path to and from the initiator. As in the first algorithm, the event *StartCut* begins the initiator's action for taking snapshot, and the algorithm also assume that the snapshot algorithm is initiated by the MSS.

The details of the Cut-Along-Tree algorithm is described by the following steps:

1. Let *T* be a *spanning tree* of the fixed network rooted at the Initiator *I*; let *parent(i)* and *children (i)* be the parent and children of node *i* in *T*.

2. The initiator executes an event *StartCut*, disables forwarding application messages to the mobile hosts under it, and then sends *Prepare* to *children (I)* and its MHs.

3. Each internal node *i*, after receiving *Prepare*, does the following:

   - send *Prepare* to all its MHs.
   - send *Prepare* to *children (i);*

4. Each MH in the cell, upon receiving *Prepare*, does the following:

   - disable the sending of application messages, record its local state,
   - send *Cut* to its MSS.
   - send its local snapshot to its MSS.

5. Each leaf node *i*, after receiving *Prepare*, does the following:

   - disable the sending of application messages (including the forwarding of application messages to its MHs), and record its local state,
   - send *Prepare* to all its MHs.
   - after receiving *Cut* from all its MHs, send *Cut* to *parent(i)*.

6. Each internal node *i*, after receiving *Cut* from *children(i)* and all its MHs, disable the sending of application messages, record its local state, and then send *Cut* to *parent(i).*

7. The initiator *I*, after receiving *Cut* from *children(I)* and from all its MHs, records its local state, and then sends *Resume* to *children(I)* and all its MHs,

8. Each internal node *i*, after receiving *Resume* from *parent(i)*, does the following:

   - send *Resume* to all its MHs
   - send *Resume* to *children (i);*
   - and then re-enable the sending of application messages.

9. Each leaf node *i*, after receiving *Resume* from *parent(i)*, send *Resume* to all its MHs, and re-enable the sending of application messages.

10. Each MH, after receiving *Resume* from its MSS, re-enable sending of application messages.

In essence, this algorithm is the optimization of the *Prepare-and-Cut* algorithm by using a pre-calculated

spanning tree and thus also obtains a consistent global state. As a result of this optimization, the message complexity of this algorithm is significantly improved. The detailed analysis is given in the remainder of this section.

Theorem 2 *The global state collected by the initiator after the Cut-Along-Tree algorithm completes is a consistent global state.*

Proof. *This algorithm also uses a two-phase approach to obtain the consistent global state. At the core of the algorithm, it is the same as the first one (Prepare-and-Cut). But, the algorithm sends out the protocol messages along a pre-calculated spanning tree, instead of by the point-to-point communication, in this sense, it is just an optimization of the first one, and this optimization has no impact on its correctness. The difference of two algorithms lies in a performance issue. Keeping this in mind, the proof that the algorithm obtains the consistent global state is very similar to the proof of* ***Theorem 1****.* ■

3.3 Analysis of Complexity: Message Efficient

These two algorithms work for network with any topology and do not require FIFO channels. Both are message efficient.

We first consider the message complexity for the *Prepare-and-Cut* algorithm. Since in this algorithm, the mobile host consumes less messages than stationary hosts (MSS), the worse case is when there is *no* mobile hosts under all MSSs. In other words, the system contains *n* processes that all are just Mobile Support Systems (MSSs) without any mobile host. In this case, there are 3 protocol messages (*Prepare*, *Cut*, and *Resume*) among the *n* processes, travelling the fixed network of *d* diameter. Therefore, the *Prepare-and-Cut* algorithm uses (*3nd)* messages in the worse case where *n* is number of the processes in the system and *d* is the diameter of the fixed network.

For the *Cut-Along-Tree* algorithm, since 3 protocol messages travel along the spanning tree, consider the worse case of a spanning tree, the algorithm uses only *3(n-1)* messages.

As a comparison, the well known Chandy-Lamport distributed snapshot algorithm [5], which assumes a completely-connected fixed network and FIFO channels, has a message complexity of $O(n^2)$. As discussed above, two algorithms use a very low message overhead that helps handle mobility (handoff).

4. Handling Mobility -- Handoff

A mobile host MH_m may move from one cell, MSS_i, to another MSS_j, during the distributed computation, care must be taken to ensure that the consistency of the global state is maintained in spite of the location change of the mobile hosts. Normally, there needs a separate protocol to *discover* that an MH has moved to a new cell, the example protocols are [8,11]; After discovery, the *handoff* begins with this MH contacting the new boss (MSS). The following *handoff* process is based on the network-layer handoff procedure [8,2].

- On discovering that it has moved from MSS_i to a new cell MSS_j, MH_m sends a *greeting*(MH_m, MSS_i) message to MSS_j,

- MSS_j acknowledges the receipt of the *greeting* message, and send *deregister*(MH_m) to MSS_i which will delete MH_m from its registration. At the same time it registers MH_m.

- If both MSS_i and MSS_j have not yet taken snapshot, or if both MSS_i and MSS_j have taken snapshot, the handoff process completes.

  If MH_m has taken its snapshot while in the cell of MSS_i and then moves to the cell of MSS_j which has not yet had snapshot. In this case, the snapshot of MH_m with the pair of $\{snp_{no}, MSS_{id}\}$ will be handed over from MSS_i, the handoff process completes. Note that MH_m will not participate in the snapshot later instructed by MSS_j as it has already had the snapshot of MH_m.

  If MH_m has not taken its snapshot while in the cell of MSS_i, and moved into the MSS_j cell which has already taken snapshot. MSS_j will instruct MH_m to take a snapshot using the *two-phase-cut* procedures, and completes the handoff process.

5. Handling Disconnection

We take a simple approach to disconnection operations of a mobile host by requiring the mobile host, MH_i, to take a (unsolicited) snapshot *prior to* disconnecting from the network as follows:

- records the local state, $local\_state_i$, of the distribute application running on MH_i,

- sends $local\_state_i$ to its local *MSS*.

Note that in a disconnected mode of operation, MH_i does not send or receive any message, and the local *MSS* will not forward any message addressed to it. Thus the local state recorded will not cause inconsistency if included in a future global state.

In responding to the snapshot request, the local *MSS* will use this pre-recorded local state as MH_i 's local snapshot which will be associated with a corresponding $\{snp_{no}, MSS_{id}\}$ pair.

6. Related Work

Mobile computing has received increasingly strong research interest. The implications of host mobility for distributed management are considered in [7], while its impact on distributed computing and fundamental challenges are considered in [16,3]. Although there were much research on consistent global states from different perspectives (for example see a collection in [17]), there are only a very few reported work considering this issue in a mobile setting [1,12,13,15]. Acharya *et al*. [1] were the first to present an asynchronous (uncoordinated) snapshot collection algorithm for distributed applications on mobile systems. Because there is no coordination for taking snapshots, an MH_i is forced to take its local snapshot whenever a message reception is preceded by a message sending at that host. This may lead to as many as local snapshots being taken as the number of application messages and thus results in a high cost. Pradhan *et al* [13] also proposed an uncoordinated protocol that considers the MH's disk storage as unstable and inappropriate for storage of a host's state. Prakash *et al.* [14,15] presented a snapshot collection algorithms for synchronously checkpointing mobile applications; it uses minimal *dependency* information to achieve low cost. Neves and Fuchs [12] describe a time coordinated recovery protocol which adapts its processing by tuning the MH's stable storage participation in the snapshot-taking process based on the current network characteristics. Our algorithms also use a coordinated approach and are very message efficient.

7. Conclusion

Consistent global states are required in a variety of distributed applications. Distributed mobile systems impose severe constraints on distributed algorithms, mainly the mobility and poor resource of mobile hosts. In this paper, we presented the *Two-Phase-Cut* algorithms for distributed mobile applications based on a two-tier system model. The algorithms rely on the resource-rich stationary hosts, which serve as a mobile support system (MSS) to guide the progress of the snapshot algorithms and maintain the consistency. These algorithms use very low message overhead to handle mobility issue, disconnection operation, and they are very message-efficient, using *O(3nd)* messages for the *Prepare-and-Cut* algorithm, where *n* is number of the processes in the system and *d* is the diameter of the fixed network, and only *3(n-1)* messages for the *Cut-Along-Tree* algorithm.

Acknowledgments

We'd like to thank anonymous referees for their instructive comments and suggestions that help improve the presentation of the paper.

12. References

1. A.Acharya and B.R. Badrinath. Checkpointing Distributed Applications on Mobile computers. In *the Proceedings of 3rd IEEE Intl. Conf. on Parallel and Distributed Information Systems*, October 1994.
2. A.Acharya and B.R. Badrinath. A Framework for Delivering Multicast Messages in Networks with Mobile Hosts. *ACM-Baltzer Journal on Mobile Networks and Applications*, 1(II):199--219, 1996.
3. B.R. Badrinath, A.Acharya, and T.Imielinsk. Impact of Mobility on Distributed Computations. *ACM Operating Systems Review*, 27(2), April 1993.
4. K.Brown and S.Singh. RelM: Reliable Multicast for Mobile Networks. Technical report, Dept of Computer Science, University of South Caroline, January 1996.
5. K.M. Chandy and L.Lamport. Distributed Snapshots: Determining Global States of Distributed Systems. *ACM Transactions on Computer Systems*, 3(1):63--75, February 1985.
6. J. M. Helary, N.Pouzeau, and M.Raynal. A Characterization of a Particular Class of Distributed Snapshots. In *Proc. of International Conf. on Computing and Information (ICCI'89)*, Toronto, Canada, May 23-27 1989.
7. T. Imielinsk and B.R. Badrinath. Wireless Mobile Computing: Challenges in Data Management. *Communications of the ACM*, 37(10):19--27, October 1994.
8. J. Ioannidis, D. Duchamp, and G. M. Jr. IP-based Protocols for Mobile Internetworking. *In Proceedings of ACM Symposium on Communication, Architectures and Protocols*, pages 235--245, September 1991.
9. J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM*

Transactions on Computer Systems, 10(1):3--25, February 1992.

10. L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558--565, July 1978.
11. T. Narten, E. Nordmark, and W. A. Simpson. Neighbor Discovery for IP Version 6 (IPv6). Internet Request for Comments RFC 1970, August 1996.
12. N. Neves and W. K Fuchs. Adaptive Recovery for Mobile Environments. *Communications of the ACM*. 40 (1):69-74, January 1997.
13. D. K. Pradhan, P.Krishna, and N.H. Vaidya. Recoverable Distributed Mobile Environments: Design and Tradeoff Issues. In the 26th International Symposium on Fault-Tolerant Computing (FTCS-26), June 1996. Also as Tech report 95-053, Texas A&M University.
14. R. Prakash, M. Raynal, and M. Singha. An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments. *Journal of Parallel and Distributed Computing*, 41:190--204, 1997.
15. R.Prakash and M.Singhal. Low-Cost Checkpointing and Failure Recovery in Mobil Computing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 7(10):1035--1048, October 1996.
16. M. Satyanarayanan. Fundamental Challenges in Mobile Computing. In *Fifteenth ACM Symposium on Principles of Distributed Computing*, Philadelphia, PA, May 1996.
17. Z. Yang and T. A. Marsland. *Global States and Time in Distributed Systems*. IEEE Computer Society Press, 1994. ISBN: 0-8186-5300-0.