Checkpointing Protocols in Distributed Systems with Mobile Hosts: a Performance Analysis

F. Quaglia, B. Ciciani, R. Baldoni

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113 - 00198 Roma, Italy
{quaglia,ciciani,baldoni}@dis.uniroma1.it

Abstract. *Checkpointing distributed applications involving mobile hosts is an important task to reduce the rollback during a recovery from a failure and to manage voluntary disconnections. In this paper we show the basic characteristics a checkpointing protocol needs to work with mobile hosts, namely, reduction of the number of checkpoints, the use of incremental checkpointing and consistent global checkpoint built on the fly. Previous points must be implemented by using as small control information as possible and ensuring little rollback. A comparative analysis of the performance of some interesting communication-induced checkpointing protocols, adapted to a mobile setting, is presented. The analysis has been carried out by using discrete event simulation and several models have been considered for the hosts mobility.*

1 Introduction

A distributed application involving mobile hosts consists of a set of cooperating processes in which some of them run on mobile hosts (MHs). An MH is connected to the wired network through a *mobile support station* (MSS). Each MSS provides a wireless coverage area to MHs called *cell*. At any given time an MH belongs, in a logical way, to only one cell, so it is connected to a *current* MSS. The current MSS acts actually as a service access point to the wired network for MHs. So, for example, each message sent by an MH, passes actually through its current MSS that provides, first, to locate the recipient of the message, then to forward the message to the current MSS of the recipient. Differently from a static host, MHs can either "roam" from one cell to another (while being active) or operate a voluntary disconnection protocol which makes them non reachable from the rest of the network until they reconnect again.

Checkpointing can be used for providing fault tolerance. In this case, upon the occurrence of a failure, a distributed application with mobile hosts should be rolled back to a consistent global checkpoint as close as possible to the end of the computation. A *global checkpoint* consists of a set of local checkpoints, one for each process, from which a distributed computation can be restarted after a failure. A *local checkpoint* is a state of a process saved onto stable storage. Informally, a global checkpoint is *consistent* if no local checkpoint in that set *happens before* [12] another one [8] (in the following we use the terms consistent global checkpoint and recovery line interchangeably). When processes take local checkpoint independently, a rollback might force the computation to its initial state (Domino Effect [15]), so a "good" checkpointing protocol should avoid that occurrence, by coordinating the action to take checkpoints in distinct processes in order that each local checkpoint is associated with a consistent global checkpoint to which it belongs (domino-free checkpointing protocols).

When developing a checkpointing protocol for mobile systems, a designer has to cope with additional issues such as vulnerability of mobile host local storage [4], voluntary disconnection, energy consumption, low bandwidth, channel contention etc. not present in conventional distributed systems [10]. For example, vulnerability of mobile host storage poses the constraint that checkpoints of a mobile host have to be transfered to the current MSS [4]. Voluntary disconnection implies that a checkpoint has to be taken each time the MH disconnects [1, 13].

In this paper we show the issues raised by a mobile setting and how they can be faced by checkpointing protocols. Then we point out the basic features a checkpointing protocol needs in order to tackle such issues. Some features, for example the use of incremental checkpointing techniques, can be implemented in any checkpointing protocol. Others allow to select some checkpointing protocols, from among those designed for distributed systems, that are well suited for a mobile setting. The selection is done according to the following criteria: (1) small number of checkpoints which decreases the number of times an MH transfer its local state to the MSS stable storage; (2) each local checkpoint should be associated on the fly with a consistent global checkpoint. Latter point avoids, in a rollback phase, heavy exchange of messages between an MH and its wired station. The previous two points should be achieved by using as small as possible control information, in order to accomplish the mobility issues, and ensuring little rollback.

Finally we consider three checkpointing protocols that match previous requirements. The two-phase-based protocol proposed by Acharya-Badrinah [1], and an adaptation to a mobile setting of the index-based protocols proposed by Briatico et al. [7] and by Quaglia et al. [14]. A performance comparison, based on the checkpointing overhead of those protocols is presented. The comparison has been carried out by taking into account different mobility and voluntary disconnection scenarios. The study of recovery schemes to associate with previous checkpointing protocols is out of the aims of this paper.

The paper is organized as follows. Section 2 lists issues raised by the mobile hosts. Section 3 presents the system model. In Section 4 we describe the two-phase-based protocol and provide a version of the protocols in [7] and [14], adapted to a mobile environment. In Section 5 the simulation study is presented.

2 Checkpointing Protocols Involving Mobile Hosts

Three classes of checkpointing protocols have been proposed for distributed systems: *uncoordinated*, *coordinated* and *communication-induced* [9]. In the first class processes take local checkpoints independently on each other and there is the risk of a domino effect while attempting to build a consistent global checkpoint during the rollback phase. In the second class, an initiator process forces other processes, during a failure-free computation, to take a local checkpoint by using control messages. The coordination can be either blocking [8] or non-blocking [11]. However, in both cases, the last local checkpoint of each process belongs to a consistent global checkpoint built on-the-fly. In the third class, the coordination is done in a lazy fashion by piggybacking control information on application messages and, usually, a local checkpoint is on the fly associated to a consistent global checkpoint.

In the following we point out constraints introduced by a mobile setting and we investigate which of previous classes can be well suited in such a setting.

2.1 Mobility Issues

When designing a protocol involving mobile hosts the following factors have to be taken into account [3, 10, 13]:

a. limited and vulnerable MH local storage;
b. low bandwidth and high channel contention;
c. voluntary disconnection/connection;
d. location cost;
e. energy consumption.

Moreover we would like to remark another issue (**f.** open system): a mobile setting is something more dynamic than a distributed environment due to mobility and connections/disconnections. So it gives rise to a scalability issue in terms of number of processes that currently carry out the application. So a good protocol should be able to add/remove processes from the application at the minimum cost.

Points (a) and (b) have been tackled by designers of protocols for systems with mobile hosts in the same way [3, 5]. Point (a) has been tackled by using a part of the stable storage of the MSS as a repository for the MH. Point (b) pushed the designers towards a client-server approach in which as many parts as possible of the checkpointing protocols are implemented by the MSS itself. In this way channel contention is reduced. Moreover also scalability can be better addressed. Other points are not yet solved in a standard way so in the following we show how they can be taken into account while designing a checkpointing protocol.

2.2 Features to Cope with Mobility

Incremental Checkpointing. In a mobile setting, each checkpointing protocol can take advantage of the use of an incremental checkpointing technique. Saving and transferring the state of a process running a mobile host on the MSS stable storage may have a large overhead in terms of battery consumption, channel utilization and so forth (points (b) and (e)). To reduce such an overhead, an incremental checkpointing technique could be used. Incremental checkpointing transfer on the MSS stable storage only the information that changed since the last checkpoint. The MSS can reconstruct the checkpoint of the process by updating its last checkpoint with the information sent by the MH. If, due to a cell switch, the last checkpoint is not present in the current MSS, the latter will incur in a transfer operation to fetch the last checkpoint from another MSS.

Number of Checkpoints and Control Information. Each time a checkpoint is taken by an MH, due to point (a), the state of an MH should be transferred to its MSS in order to make it retrievable at any time upon the request of a global checkpoint collection.

So, a good checkpointing protocol involving mobile hosts should reduce as many as possible the number of checkpoints in order to achieve a reduction of the energy consumption and channel contention. Due to points (b), (d) and (e), the reduction of the number of checkpoints should be realized by keeping as small as possible either the number of control messages (in the case of the coordinated approach) or the amount of piggybacked control information (in the case of communication-induced approach).

Consistent Checkpoints Built On-The-Fly. When a failure occurs an application is required to rollback to a consistent global checkpoint. In a mobile setting, it would be convenient to rollback by using as few messages as possible. An heavy exchange of messages to establish a consistent global checkpoint could consume many resources such as MH energy, channels' bandwidth, etc. Checkpointing protocols that ensure a light rollback phase well tackle points (b), (d) and (e).

Global Checkpoint Collection Latency. Connections and disconnections may significantly increase the completion time of the construction of a consistent global checkpoint. Since an MH cannot provide its local checkpoint while being disconnected, a local checkpoint has to be taken each time an MH disconnect from the network (thus the checkpoint taken upon disconnecting will belong to every global consistent checkpoint of the application collected during the disconnection period of the MH) [13].

From the previous paragraphs, it comes out that uncoordinated protocols are not suitable to mobile settings as, in case of failure, a huge amount of messages has to be exchanged among all the hosts to build a consistent global checkpoint. Remark that the mobile setting increases the number of potential failures compared to a wired distributed system.

To show advantages and drawbacks of a coordinated checkpointing protocol in a mobile setting, let us consider the Chandy-Lamport protocol [8]. The advantage of this protocol lies in its simplicity of implementation. When a consistent checkpoint has to be build, an initiator sends a control message (the *marker*) to each participant of the distributed computation. All messages sent before (resp. after) the marker by the initiator to a process are delivered before (resp. after) the marker. The arrival of the marker forces a process to take a local checkpoint. However, if mobile hosts are included in the computation, some drawbacks arise. (1) As mobile hosts have not a static location, this might imply one search cost - point (d) - per mobile host just to deliver a control message. (2) The introduction of control messages increases the channel contention in a cell and drains the MH battery - points (a), (b) and (e) -. (3) The computation does not scale when new processes are added - point (f) -. (4) As checkpointing is a very high-cost operation, only the MHs that really need to take it should be forced. For example, hosts that do not send/receive messages for a while might not be required to take a checkpoint.

The Prakash-Singhal protocol [13] answers to point (4) by executing an explicit coordination among a subset of processes. This subset contains the processes which have established causal dependencies since the last coordination. However, as the protocol adds control messages and lies on data structures whose logical size is the number of processes of the distributed computation, points (1), (2), and (3) remain, at least partially, unanswered.

As far as communication-induced checkpointing is concerned, two interesting aspects have to be remarked: the answer to points (1) and (2) comes directly from the lazy coordination approach (the control information is routed together with the application information). Moreover, only an MH that receives an application message might be directed to take a checkpoint (this answers to point (4)).

Hence, the communication-induced checkpointing protocols seems to be the family that can be better adapted to a mobile setting. In Section 4 we describe the Acharya-Badrinah protocol and the protocols proposed by Briatico-Ciuffoletti-Simoncini and Quaglia-Baldoni-Ciciani. We will show how these protocols differ when answering to point (3).

3 System Model

We consider a mobile environment consisting of n MHs $(h_1, h_2, \ldots h_n)$ and r MSSs. MHs are *autonomous* in the sense that: they do not share memory, and do not have access to private information of other MHs such as clock speed, etc.

MHs interact solely by messages exchanging; we assume a reliable communication subsystem that ensures an *at-least-once* message delivery semantic (see [2] for transport protocols that ensure this property) and message transmission takes an unpredictable but finite amount of time.

An MH produces a sequence of events which result by the execution of internal, send or receive statements. Send and receive events of a message m are denoted respectively by *send*(m) and *receive*(m). Each event moves the MH from one state to another. A local checkpoint dumps the current MH state onto stable storage. As remarked in the previous sections, due to the vulnerability of MH local storage, the MH checkpoint is transferred to its current MSS.

A checkpoint of a mobile host h_i is denoted as $C_{i,x}$ where x is called the index of the checkpoint. A message m sent by h_i to h_j is called *orphan* with respect to a pair (C_{i,x_i}, C_{j,x_j}) iff its receive event occurred before C_{j,x_j} while its send event occurred after C_{i,x_i}. A *global checkpoint* $\mathcal{C}$ is a set of local checkpoints $(C_{1,x_1}, C_{2,x_2}, \ldots, C_{n,x_n})$ one for each MH. A global checkpoint $\mathcal{C}$ is *consistent* if no orphan message exists in any pair of local checkpoints belonging to $\mathcal{C}$ [8].

Due to the nature of the mobile computing environment, an MH takes a local checkpoint either when it switches from one cell to another, or upon disconnecting from the network. These checkpoints cannot be avoided by an MH and they will be referred to as *basic* checkpoints. An MH takes also additional checkpoints induced by some communication patterns (*forced* checkpoints) in order to have at any time a consistent global checkpoint.

4 Communication-Induced Protocols with Mobile Hosts

4.1 Two-Phase-Based Protocol

The two-phase-based protocol (TP) from Acharya-Badrinath [1] actually adapts the Russel's protocol [16] to the context of mobile systems. Each MH h_i owns a boolean variable *phase*$_i$ which can assume only two values (SEND and RECV); upon receiving a message, if the value of *phase*$_i$ is SEND then h_i takes a checkpoint (*forced* checkpoint) and *phase*$_i$ is set to RECV; *phase*$_i$ is set to SEND each time h_i sends a message. Below we show the checkpointing procedures executed at each MH.

Procedures executed at an MH h_i

```
Procedure init:
  $phase_i := RECV$;
  execute checkpointing procedure; % basic checkpoint %

Procedure send $m$ to $h_j$:
  send(m) to $h_j$;
  $phase_i := SEND$;

Upon the receipt of a message $m$
  If $phase_i = SEND$ then
  begin
     execute checkpointing procedure; % forced checkpoint %
     $phase_i := RECV$
  end;

When switching cells:
  execute checkpointing procedure; % basic checkpoint %

When disconnecting from the network:
  execute checkpointing procedure; % basic checkpoint %

Checkpointing procedure:
  take a checkpoint C;
```

Acharya and Badrinath proved that to keep track of the consistent global checkpoint a local checkpoint belongs to, in this protocol, a vector of integers must be piggybacked on each application message, that takes into account the causal dependency established between local checkpoints. For this reason, in the

TP protocol each MH h_i piggybacks on every application message two vectors: $CKPT_i[]$ and $LOC_i[]$. $CKPT_i[]$ is a transitive dependency vector on checkpoint intervals, $LOC_i[]$ is a transitive dependency vector on MH locations. Upon taking a checkpoint, both these vectors are recorded on stable storage. The vector $LOC_i[]$ is used for an efficient retrieval of checkpoints over the wired network as follows: if $CKPT_i[j] = p$ and $LOC_i[j] = q$, then a global checkpoint that includes the $CKPT_i[i]$-th checkpoint of h_i must include the p-th checkpoint of h_j which is located at the q-th MSS.

As to build a consistent global checkpoint on the fly the TP protocol piggybacks one vector of integers (whose size is given by the number of hosts), the TP protocol does not scale while changing the number of hosts.

4.2 Index-Based Protocols

In these protocols each hosts h_i assigns a sequence number sn_i to each local checkpoint. It is assumed to assign sn_i equal to zero to its first checkpoint. The number sn_i is attached as a control information $m.sn$ on each outgoing message m. Those protocols enforce consistency among checkpoints with the same sequence number.

Briatico-Ciuffoletti-Simoncini's Protocol. In the checkpointing protocol proposed by Briatico, Ciuffoletti and Simoncini (BCS) [7], a forced checkpoint is taken whenever a message m such that $m.sn > sn_i$ is received. The procedures for updating sequence numbers and taking checkpoints in a mobile setting are the following:

Procedures executed at an MH h_i

```
Procedure init:
  $sn_i := 0$;
  execute checkpointing procedure; % basic checkpoint %

When sending a message $m$ to $h_j$:
  $m.sn := sn_i$;
  send(m) to $h_j$;

Upon the receipt of a message $m$:
  If $m.sn > sn_i$ then
  begin
     $sn_i := m.sn$;
     execute checkpointing procedure % forced checkpoint %
  end;

When switching cell:
  $sn_i := sn_i + 1$;
  execute checkpointing procedure; % basic checkpoint %

When disconnecting from the network:
  $sn_i := sn_i + 1$;
  execute checkpointing procedure; % basic checkpoint %

Checkpointing procedure:
  take a checkpoint $C_{i,sn_i}$;
```

It has been proved in [7] that a consistent global checkpoint includes local checkpoints with the same sequence number, one for each process (if there is a jump in the sequence number of a process, the first checkpoint with greater sequence number must be included).

As the sequence number is actually a replicated data whose value is shared by the hosts of the computation, the BCS protocol scales well with respect to the number of hosts.

Quaglia-Baldoni-Ciciani's Protocol. The checkpointing protocol proposed by Quaglia, Baldoni and Ciciani (QBC) in [14] is an optimization of BCS. It reduces differences between sequence numbers of distinct MHs. As shown in [6, 14], this leads to a reduction of the number of forced checkpoints.

Each MH h_i endows a receive number rn_i which records the maximum sequence number received by h_i with application messages, as well as a sequence number sn_i. The receive number rn_i is used for detecting whether a checkpoint of h_i can replace its predecessor (in the same host) in the recovery line. At the time of taking a basic checkpoint C, if $rn_i < sn_i$, then C does not depend on any checkpoint in the recovery line with sequence number sn_i. In this case, its sequence number is set to sn_i, and the checkpoint C replaces its predecessor in the recovery line with sequence number sn_i. The capability to replace a checkpoint with another one in a recovery line has been formalized in [6, 14] by the introduction of an equivalence relation between local checkpoints. Such a relation states that two local checkpoints A and B of h_i are equivalent with respect to the recovery line $\mathcal{L}$, if A belongs to $\mathcal{L}$, and the set $\mathcal{L}$-$\{A\}\cup\{B\}$ is a recovery line.

QBC leads to a slower increase of sequence numbers of MHs compared to BCS with the advantage of less induced checkpoints. The procedures for updating sequence, receive numbers and taking checkpoints are the following:

Procedures executed at an MH h_i

```
Procedure init:
  $sn_i := 0$;
  $rn_i := -1$;
  execute checkpointing procedure; % basic checkpoint %

When sending a message $m$ to $h_j$:
  $m.sn := sn_i$;
  send(m) to $h_j$;

Upon the receipt of a message $m$:
  $rn_i := max(m.sn, rn_i)$;
  If $m.sn > sn_i$ then
  begin
    $sn_i := m.sn$;
    execute checkpointing procedure % forced checkpoint %
  end;

When switching cell:
  If $rn_i = sn_i$ then $sn_i := sn_i + 1$;
  execute checkpointing procedure; % basic checkpoint %
```

```
When disconnecting from the network:
  **If** $rn_i = sn_i$ **then** $sn_i := sn_i + 1$;
  execute checkpointing procedure; % basic checkpoint %

Checkpointing procedure:
  take a checkpoint $C_{i,sn_i}$;
```

A consistent global checkpoint can be built by using the same rule of the BCS protocol [6, 14], and QBC scales well as BCS.

5 A Performance Study

5.1 Simulation Model

This section presents simulation results of a performance study to compare the checkpointing overhead of TP, BCS, and QBC. The simulation has been carried out in a mobile environment with 5 MSSs and 10 MHs. Each MH, while being active, can send a message to any other and the destination of each message is a uniformly distributed random variable.

The execution time for an internal event of an MH is exponentially distributed with mean 1.0 time units. Whenever an MH communicates, it executes a send or a receive operation with probability P_s and $1 - P_s$, respectively. The sending and the receiving of a message over the wireless cell and the message transfer between adjacent MSSs takes 0.01 time units.

The hand-off protocol consists of sending two messages one to the MSS which is going to be leaved, the other to the MSS which is going to be current to the MH. The disconnection protocol consists of sending a message to the current MSS.

Upon entering a new cell, an MH with probability P_{switch} will switch to another cell (after an exponentially distributed period of time with mean T_{switch}), while, with probability $(1 - P_{switch})$ it will disconnect from the network (after an exponentially distributed period of time with mean $T_{switch}/3$). The time between disconnecting and reconnecting is exponentially distributed with mean 1000 time units.

We also consider a degree of heterogeneity (H) among MHs. For example, $H = 0\%$ means all MHs have the same average time T_{switch} of permanence in a cell; $H = 25\%$ (resp. $H = 75\%$) means 25% (resp. 75%) of MHs has an average permanence time $T'_{switch} = T_{switch}/10$ while the remaining 75% (resp. 25%) has an average permanence time T_{switch}. This is done in order to take into account the different mobility features of the involved MHs.

A first series of experiments were conducted with $H = 0\%$ and by varying T_{switch} from 100 to 10000 time units and we measured the number N_{tot} of checkpoints taken by a protocol. In a second series of experiments we varied the degree of heterogeneity H of the MHs.

As we are interested only in counting how many local states of the MHs are recorded as checkpoints, the time for taking a checkpoint is not considered (i.e., the checkpoint insertion is considered as instantaneous), however, we simulated situations in which the time for taking a checkpoint is non negligible and we did not found a remarkable impact on the number of taken checkpoints. Each run simulates 100000 time units and for each value of T_{switch} and H, we did several simulation runs with different seeds and the result were within 4% of each other, thus, variance is not reported in the plots.

5.2 Results of the Experiments

Figures 1 and Figure 2 show N_{tot} as a function of T_{switch} for environments with $H = 0\%$ and with $P_{switch} = 1.0$ (MHs never disconnect) and $P_{switch} = 0.8$, respectively.

The results show that both BCS and QBC perform better than TP in terms of total number of checkpoints imposed to the execution (with a gain up to 90% when $T_{switch} = 10000$ time units). Furthermore, QBC gains over BCS especially when considering the environment (see Figure 2) in which MHs disconnect from the network (the gain is up to 15% in such an environment).

QBC also shows good performance in heterogeneous environments (see Figures 3, 4, 5 and 6) in which there are some MHs which take basic checkpoints more frequently than others (due to their short average time of permanence in a cell).

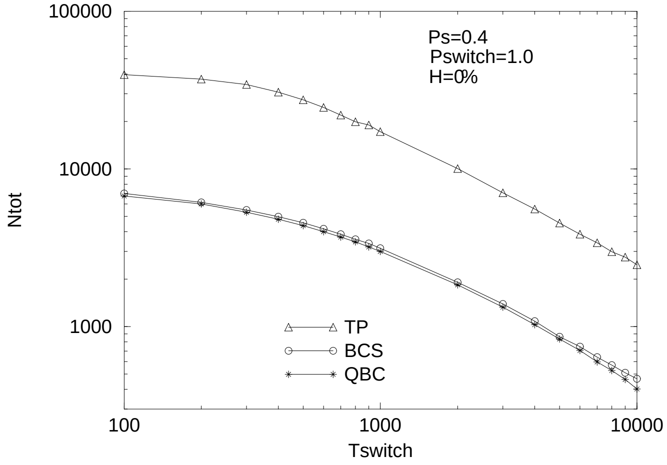

Fig. 1. N_{tot} vs. the average time of permanence in a cell T_{switch}, in a homogeneous environment with $P_s = 0.4$ and $P_{switch} = 1.0$ (MHs never disconnect from the network).

In these environments, BCS pushes the sequence numbers of MHs to diverge, thus the gain achievable by QBC is further amplified with respect to the homo-

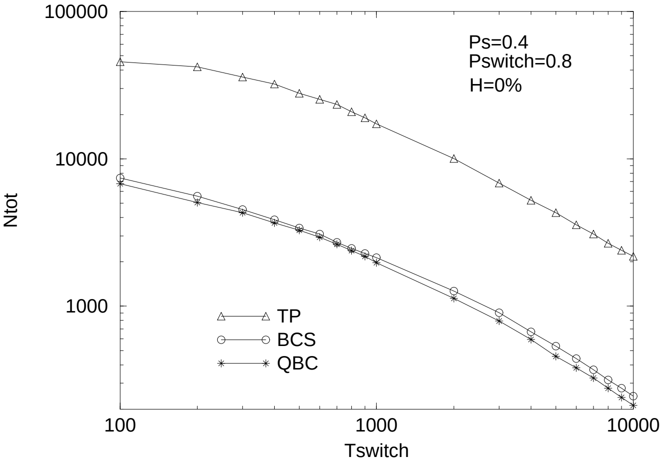

Fig. 2. N_{tot} vs. the average time of permanence in a cell T_{switch}, in a homogeneous environment with $P_s = 0.4$ and $P_{switch} = 0.8$.

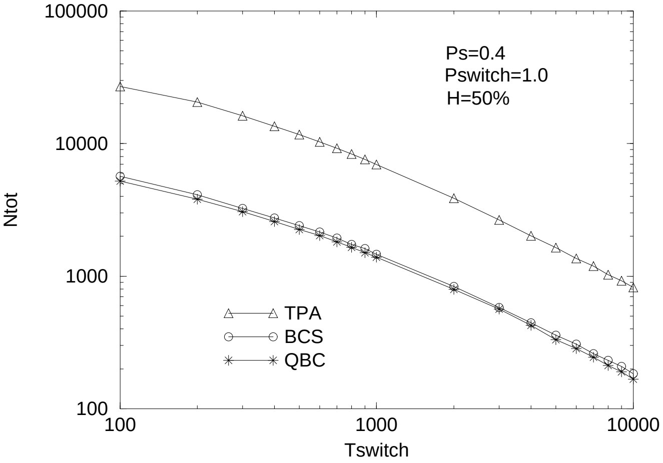

Fig. 3. N_{tot} vs. the average time of permanence in a cell T_{switch} of the slowest MHs in a heterogeneous environment ($H = 50\%$) with $P_s = 0.4$ and $P_{switch} = 1.0$

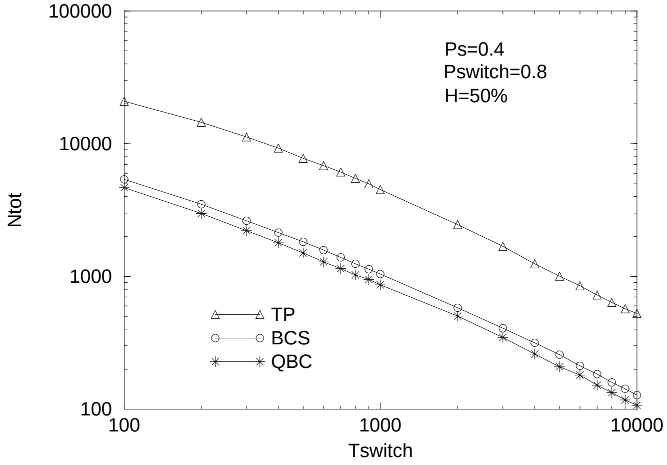

Fig. 4. N_{tot} vs. the average time of permanence in a cell T_{switch} of the slowest MHs in a heterogeneous environment ($H = 50\%$) with $P_s = 0.4$ and $P_{switch} = 0.8$.

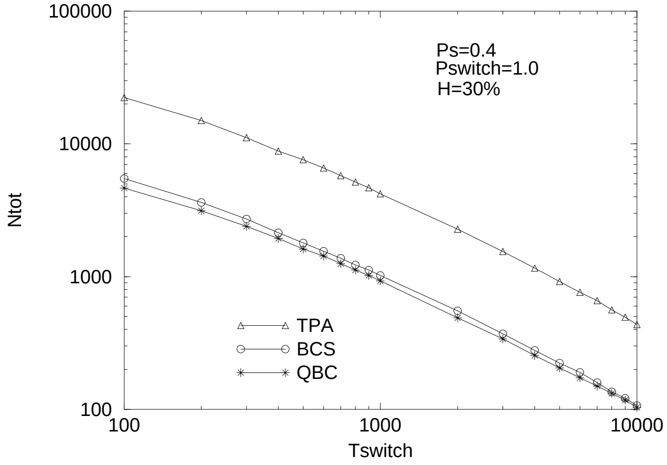

Fig. 5. N_{tot} vs. the average time of permanence in a cell T_{switch} of the slowest MHs in a heterogeneous environment ($H = 30\%$) with $P_s = 0.4$ and $P_{switch} = 1.0$.

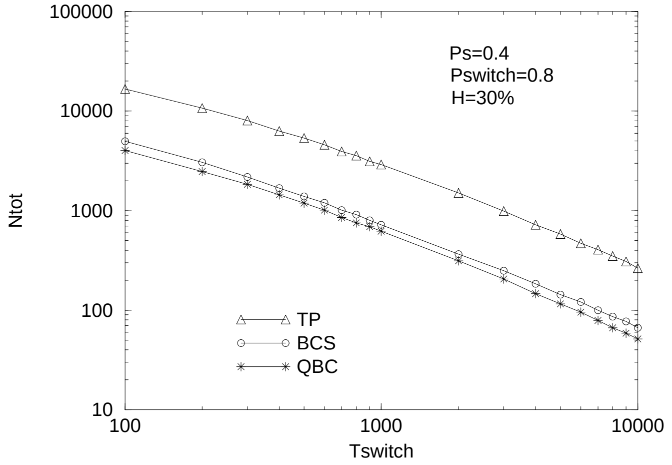

Fig. 6. N_{tot} vs. the average time of permanence in a cell T_{switch} of the slowest MHs in a heterogeneous environment ($H = 30\%$) with $P_s = 0.4$ and $P_{switch} = 0.8$.

geneous environment. In particular, the maximum gain of QBC over BCS (up to 23%) is achieved in the case of $H = 30\%$, $P_{switch} = 0.8$ and $T_{switch} = 10000$ time units. The results show that index-based protocols perform better than the two-phase-based one in all the simulated environments (thus, independently on both the mobility characteristics and the disconnection rate of MHs). Furthermore, the QBC protocol outperforms both the other protocols, especially in heterogeneous environments, where the reduction of the difference between checkpoint sequence numbers lead to a strong reduction of the number of forced checkpoints.

6 Conclusion

In this paper we present a short overview of problems introduced by checkpointing in distributed systems with mobile hosts. We describe how different checkpointing approaches cope with this type of environment and then we presented a comparative analysis of several particularly interesting communication-induced checkpointing protocols, working in a mobile computing system. Such a comparison has been carried out by a simulation study while varying both the mobility assumptions and the disconnection rate of the mobile hosts. We simulated also heterogeneous environments to point out the performance of the protocols in a broad variety of scenarios. The results show that index-based protocols perform better than the two-phase one and well address the scalability issue of a mobile setting. Among the index-based protocols proposed, the QBC protocol shows the best performance due to the reduction of the differences between sequence numbers in different mobile hosts, which is obtained without adding control information. Such a reduction gets larger in heterogeneous environments, which actually model more realistically a mobile computing environment.

Future work is focused on the evaluation of the recovery time and of the amount of undone computation due to a failure.

References

1. Acharya, A. and Badrinath, B. R., Checkpointing Distributed Application on Mobile Computers, *Proc. 3-th International Conference on Parallel and Distributed Information Systems*, 1994.
2. Acharya, A. and Badrinath, B. R., Delivering Multicast Messages in Network with Mobile Hosts, *Proc. 13-th International Conference on Distributed Computing Systems*, 1993.
3. Alagar, S. and Venkatesan, S., Causal Ordering in Distributed Mobile Systems, *IEEE Trans. on Computers*, 46(3): 353-361, 1997.
4. Alonso, R. and Korth, H., Database Systems Issues in Nomadic Computing, *Proc. ACM SIGMOD International Conference on Management of Data*, 1993.
5. Badrinath, B. R., Acharya, A. and Imielinsky, T., Structuring Distributed Algorithms for Mobile Hosts, *Proc. 14-th International Conference on Distributed Computing Systems*, 1994.
6. Baldoni, R., Quaglia, F. and Fornara, P., An Index-Based Checkpointing Algorithm for Autonomous Distributed Systems, *Proc. 16-th IEEE Int. Symposium on Reliable Distributed Systems*, 1997.
7. Briatico, D., Ciuffoletti, A. and Simoncini, L., A Distributed Domino-Effect Free Recovery Algorithm, *in Proc. IEEE Int. Symposium on Reliability Distributed Software and Database*, 1984.
8. Chandy, K.M. and Lamport, L., Distributed Snapshots: Determining Global States of Distributed Systems, *ACM Transactions on Computer Systems*, 3(1): 63-75, 1985.
9. Elnozahy, E. N., Johnson, D. B. and Wang, Y. M., A Survey of Rollback-Recovery Protocols in Message-Passing Systems, *Technical Report* CMU-CS-96-181, Carnegie-Mellon University, 1996.
10. Imielinsky, T. and Badrinath, B. R., Wireless Computing, *Communications of the ACM*, 37(10): 19-27, 1994.
11. Koo, R. and Toueg, S., Checkpointing and Rollback-Recovery for Distributed Systems, *IEEE Transactions on Software Engineering*, 13(1): 23-31, 1987.
12. Lamport, L., Time, Clocks and the Ordering of Events in a Distributed System, *Communications of the ACM*, 21(7): 558-565, 1978.
13. Prakash, R. and Singhal, M., A Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems, *IEEE Transactions on Parallel and Distributed Systems*, 7(10): 1035-1048, 1996.
14. Quaglia, F., Baldoni, R. and Ciciani, B., A Checkpointing-Recovery Scheme For Domino Free Distributed Systems, *Proc. 2-nd Workshop on Fault Tolerant Parallel and Distributed Systems*, 1997.
15. Randell, B., System structure for software fault tolerance, *IEEE Transactions on Software Engineering*, SE1(2):220-232, 1975.
16. Russell, D.L., State Restoration in Systems of Communicating Processes, *IEEE Transactions on Software Engineering*, SE6(2): 183-194, 1980.

This article was processed using the LaTeX macro package with LLNCS style