Distrib Parallel Databases (2008) 23: 189–205
DOI 10.1007/s10619-008-7026-3

Movement-based checkpointing and logging for failure recovery of database applications in mobile environments

Sapna E. George · Ing-Ray Chen

Published online: 8 March 2008

Abstract In this paper, we present an efficient failure recovery scheme for mobile database applications based on movement-based checkpointing and logging. Current approaches take checkpoints periodically without regard to the mobility behavior of mobile users. Our movement-based checkpointing scheme takes a checkpoint only after a threshold of mobility handoffs has been exceeded. The optimal threshold is governed by the failure rate, log arrival rate, and the mobility rate of the mobile host. This allows the tuning of the checkpointing rate on a per-user basis. We identify the optimal movement threshold which will minimize the recovery cost per failure as a function of the mobile node's mobility rate, failure rate and log arrival rate. We derive the mobile database application recoverability, i.e., the probability that the recovery can be done by a specified recovery time deadline. Numeric data are presented to demonstrate the feasibility of our approach with its applicability given.

Keywords Mobile database application · Mobile data management · Failure recovery · Checkpoint · Logging · Recoverability · Mobility handoff

1 Introduction

Advancement in wireless networking and portable devices is revolutionizing the way individuals and businesses view computing. Many industries are now trying to provide services to this market and mobile applications are expected to become the norm

Recommended by Ahmed K. Elmagarmid.

S.E. George · I.-R. Chen (✉)
Department of Computer Science, Virginia Polytechnic and State University, Falls Church, VA 22043, USA
e-mail: irchen@cs.vt.edu

S.E. George
e-mail: sgeorge@vt.edu

in the near future. However, certain inherent properties of mobile computing—a type of distributed computing involving hosts that may be mobile while retaining network connectivity through wireless communications—such as host mobility, disconnections, wireless bandwidth limitations, makes these applications susceptible to failures typically not encountered in the traditional computing environments. This paper concerns failure recovery of mobile database applications.

Mobile database application recovery is different from database transaction recovery since an application may involve multiple database transactions, multiple states and it lacks a formal definition of application state similar to the mathematical foundations of database transactions [5]. Typically, distributed systems achieve fault-tolerance through schemes such as checkpointing, logging and rollback recovery [3]. During failure-free operation, processes save their state to a stable storage periodically, called checkpoints. Upon failure, a failed process recovers by rolling back to the latest checkpoint and restarting computation from this intermediate state. However, inter-process dependencies may result in cascading rollbacks, which in the extreme case may take the system all the way back to its initial state, often termed the domino effect. Asynchronous recovery of a failed process is achieved by combining checkpointing with logging, where all the non-deterministic events that a process executes as well as the information necessary to replay these events are logged to the stable storage in addition to the checkpoints. During recovery, the failed process rolls back to the latest checkpoint and replays all the logged events in their original order, there by recreating its pre-failure state independently.

Many approaches have been proposed that refines these basic mechanisms for improvements in performance and cost during failure-free operations and recovery [8]. In spite of that, mobile computing introduces new challenges that preclude the direct application of these mechanisms to mobile distributed systems. The properties of the mobile computing environment that drives the rethinking of failure recovery mechanisms are mobility of hosts, low bandwidth and unreliable network connectivity, limited battery life of host devices, lack of stable storage on host devices, and different types of failures—voluntary disconnection, long term hardware failures, and short-term software failures.

Due to these characteristics, traditional recovery schemes suffer from many shortcomings when applied to the mobile computing environment. At the same time, the failure-prone nature of the environment makes it essential to provide some form of explicit recovery mechanism. In light of the above discussion, this paper presents a novel movement-based checkpointing strategy combined with logging for recovery of individual hosts in mobile computing environments. The approach takes checkpoints after a certain number of host migrations across cells rather than periodically. This movement threshold is a function of failure rate, log arrival rate, and the mobility rate of the application and the mobile host (MH), which allows adaptation to user and application behavior. To the best of our knowledge, none of the existing algorithms consider the effects of mobility on checkpoint intervals, but rather assume periodic checkpointing. Mobility is factored only into the management of recovery information such as when to consolidate logs dispersed across many mobile support stations (MSSs) [10]. The performance of the proposed scheme is evaluated through analytic methods and results are presented later in the paper. In addition, this paper

demonstrates that there exist optimal movement thresholds depending on operating conditions and recovery deadline.

The proposed movement-based checkpointing and recovery algorithm is suitable to a wide range of database applications in mobile environments since it considers application/user behavior as defined by its recovery deadline and log arrival rate, nature of the mobile environment defined by failure rate and MH behavior defined by its mobility rate, in determining the optimum movement threshold to trigger checkpointing. As an example, consider a mission critical application providing communications and shared situational awareness to an active military unit. The failure rate in such a mobile environment is likely to be very high; mobility rate of users is also likely to be high. At the same time, fast recovery from failures is more important than minimizing cost of failure-free operations. Note that recovery time is governed by how far away the last checkpoint is located from the MSS in which a host recovers, how many log entries must be transferred to the recovery MSS, how dispersed these logs are, and the time it takes to load and execute them at the MH. The proposed scheme allows the dynamic determination of an optimum movement threshold, and hence checkpointing rate, that will minimize recovery time while balancing cost of recovery. This optimum threshold will ensure that the checkpoints stay close to the recovery MSS and that the logs are not too widely dispersed. In contrast, schemes that employ constant periodic checkpointing do not consider the effect of user mobility on checkpointing. If the chosen checkpoint rate is too low, the last checkpoint may be located very far from the recovery MSS and there may be a large number of logs dispersed across many MSS resulting in a large recovery time. If the rate is too high, precious wireless resources may be unnecessarily consumed in taking and managing checkpoints.

The rest of this paper is organized as follows. Section 2 surveys related work and highlights differences from our approach. Section 3 describes the mobile computing system assumed in this paper. Section 4 elaborates the proposed movement-based checkpointing, logging and recovery mechanism. Section 5 shows the modeling and performance analysis of the proposed scheme along with mathematical formulations of relevant performance parameters such as time required for failure recovery and total recovery cost. Finally, Section 6 summarizes the paper, discusses the applicability, and outlines future research areas.

2 Related work

Application failure recovery in the mobile computing environment has received considerable attention in the recent years. The schemes that have been proposed employ checkpointing, logging or a combination of both, recognizing the inherent limitations of the mobile computing environment.

Acharya et al. in [1] describes a distributed uncoordinated checkpointing scheme, where multiple MHs can arrive at a global consistent checkpoint without coordination messages. However, this paper does not describe how failure recovery is achieved nor does it address the issue of recovery information management in the face of MH movement. Neves and Fuchs [7] also proposed a checkpointing only scheme that achieves global consistent checkpoint without additional messaging but is unique in

that it uses time to synchronize checkpoint creation. Pradhan, Krishna, and Vaidya [11] proposed a recovery scheme that combines various checkpointing and logging schemes for different mobile environments. They describe two uncoordinated checkpoint protocols, no-logging and logging and three strategies for recovery information management due to MH mobility: pessimistic, lazy and trickle strategies.

T. Park et al. in [10] proposed a recovery mechanism that enables independent recovery by MHs by employing periodic checkpointing and a combination of pessimistic and optimistic logging. The main feature of this paper that we have adapted in our scheme is the notion of movement-based management of recovery information. In their approach, checkpoints are triggered periodically and when a MH moves outside of a certain range, the recovery information is transferred reactively to the local MSS. In contrast to [10] and [11], our approach considers a MH's movement pattern in the checkpointing strategy in order to avoid costly transfers of all recovery information. In [9], T. Park et al. proposed an asynchronous recovery scheme using checkpointing and logging. However, in this paper, they also consider the case of unreliable MSSs where the recovery information of a MH may be lost due to failure of a MSS. In this case, in order to enable consistent recovery, every other dependent MHs must be traced and rolled back.

Yao, Ssu and Fuchs [13] proposed an algorithm that combines checkpointing and message logging in the Mobile IP environment. Its components execute on the MSS, the Home Agent (HA) and the MH. All messages are logged and the MH takes checkpoints periodically and stored at the MSS that is serving as the current Foreign Agent. When a MH leaves a cell, the old MSS informs the HA of the ids of the checkpoint and logs stored at the MSS. Thus, the HA maintains the latest itinerary of the MH and can be queried upon recovery to collect the distributed recovery information. Higake and Takizawa [6] proposed a hybrid checkpoint recovery scheme. A mobile host leaves an agent on every MSS in its itinerary. During recovery processes MHs roll back to a consistent state with the help of these agent processes. Chen et al. [2] considered recoverability issue of mobile applications with periodic checkpointing. Our paper extends the analysis to mobility-based checkpointing and logging for efficient fault recovery of database applications in mobile environments.

3 Mobile computing system

The mobile computing system assumed in this paper follows the model presented in [1]. The system consists of a set of mobile hosts (MHs) that are free to move around. At any time, they maintain network connectivity through a wireless link to a static mobile support station (MSS). The MSSs are interconnected through high speed static wired networks. A MSS handles all communications to and from MHs within its area of influence known as a *cell* usually determined by the range of wireless transmission. Assuming a hexagonal shape for each cell, a hexagonal network coverage model will be formed by a community of cells. Thus, sending a message to another MH consists of two one-hop wireless transmissions between the sender and receiver MHs and their respective local MSSs in addition to an arbitrary number of hops across the wired infrastructure between the sender's MSS and receiver's MSS.

The wired and wireless network protocols are assumed to provide reliable FIFO delivery of messages with arbitrary delay to the application.

Considering that the MH's disk cannot be assumed to be stable, each MSS is equipped with enough volume of stable storage to store the state and log information for all the MHs currently in its cell as well as those that were recently in its cell. However, due to the fact that MSSs must support multiple concurrent MHs, this storage must be efficiently managed.

The interactions between the MH and the network infrastructure most relevant to failure recovery are handoff, disconnect and reconnect. When a MH crosses a cell boundary due to mobility, it first establishes connection with the new MSS in the new cell giving the MSS its ID and the ID of the previous MSS. It then disconnects from the previous MSS. This process usually occurs instantaneously and is called a handoff. A MH may also disconnect voluntarily from the network to conserve power and reconnect at a later time. Due to mobility of hosts, it is common for a MH to disconnect in one MSS and reconnect in another. In this case, the MH sends the ID of the previous MSS to the new MSS, which in turn initiates proper handoff procedures.

In this paper a distributed computation is assumed to consist of a number of processes executing concurrently on multiple hosts. A single process may be in either one of three states at any point of time: normal, save or recovery. In the normal state, it may be executing application related computations, receiving user inputs or sending and receiving messages. Occasionally, each process saves its state as a checkpoint to the stable storage (save state). During this operation, the MH stops execution and compiles the current values of all non-transient state variables into a message, assigns it a unique id and sends it to the MSS for storage. Between checkpoints the application performs logging activities to record incremental state changes. The application's logging behavior assumed in this paper follows the model described in [1]. Application state may change due to receipt of messages or due to user inputs. These are commonly referred to as 'write events'. If the write event is a message received from another MH or a server, the MSS first receives it, logs it to stable storage and then forwards it to the MH. Thus no overhead is incurred during this process. On the other hand, if the write event is a user input or a local computation, the MH first forwards a copy to the MSS to be logged and does not apply it locally until an acknowledgement is received from the MSS. Thus, logging is also an activity that the MH and MSS execute during normal operations. Every fresh checkpoint purges old checkpoints and logs, possibly distributed over many MSS. More details of recovery information management in provided in Section 4.3.

4 Movement-based checkpointing and logging

The recovery scheme presented here combines independent checkpointing and pessimistic message logging enabling asynchronous recovery of a MH upon failure. In general application recovery mechanisms try to optimize recovery cost (failure-free operational cost), recovery time and storage requirements for recovery related information. The general approach taken by current schemes is to create checkpoints periodically (possibly based on an application parameter setting such as maximum

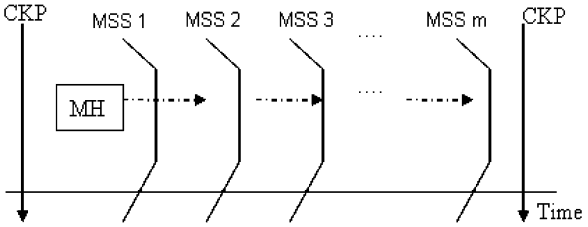

Fig. 1 Movement-based checkpointing

time to recover or failure rate) and subsequently control the proliferation of recovery information using techniques that merge logs and move the information closer to the MH. One such scheme uses distance or number of handoffs as the parameter that triggers information consolidation [10]. In this approach, when the MH crosses a distance threshold from the location of the latest checkpoint, the recovery information is collected and transferred to the MH's local MSS.

In contrast, the recovery protocol described here proactively controls the number of checkpoints and logs by using a movement-based checkpointing strategy. This means that the additional overhead of unnecessary checkpoints and log consolidation during failure-free operation is avoided.

4.1 Checkpointing and message logging

Each mobile host independently takes checkpoints of the application state and between checkpoints all write events are logged at the current MSSs' stable storage. The interval between checkpoints is governed by the number of handoffs experienced by the MH and is not fixed. Each MH maintains a *handoff_counter* which is incremented by 1 every time a handoff occurs. When the value of the counter becomes greater than a threshold M, a checkpoint is taken. The process is illustrated in Fig. 1.

The value of M is a function of the user's mobility rate, the failure rate and log arrival rate (as shown in Section 5.1). The host assigns a unique sequence number to the checkpoint and sends it to the MSS which saves it to stable storage. The MH maintains two variables locally related to checkpoints: *cp_seq* which stores the sequence number of the latest checkpoint and *cp_loc* which stores the ID of the MSS that has recorded the latest checkpoint. Let this MSS be called MSS_{cp}. After taking the checkpoint, the MH resets the *handoff_counter* to 0. Thus, depending on the variability in the MH's mobility, the time interval between successive checkpoints differs. In between checkpoints, all write events related to a MH is also logged to the local MSS. Each log entry is time stamped so that they can be replayed in the correct order during recovery. The MH locally maintains a list *log_set* containing the IDs of MSSs that stores its logs. Let this set of MSSs be called MSS_{logs}.

At every checkpoint, *cp_loc* is updated with the current MSS, *cp_seq* is updated with the sequence number of the latest checkpoint and *log_set* is cleared. At every logging activity, the ID of the current MSS is added to *log_set* if it is not present already. The performance of this scheme depends on identifying the optimal movement threshold M per user and application. This value ensures that the checkpoints and logs remain within acceptable range of the MH's current location thereby eliminating the need for information consolidation. In addition, it also ensures acceptable recovery time since M bounds the number of MSSs' from which logs must be retrieved.

4.2 Independent recovery

The checkpoints and the logs enable a MH to recover independently without requiring coordination with other hosts. This paper makes no assumption that the MH must recover in the same MSS in which it failed. In order to perform rollback recovery, after the MH reconnects to a MSS after failure, it sends to the current MSS the sequence number of the latest checkpoint and the ID of the MSS storing it. Recall that these values are stored locally at the MH in the variables *cp_seq* and *cp_loc*. The MSS initiates the process of collecting the checkpoint and the logs. For the former, it sends a request with the checkpoint sequence number to the MSS holding the checkpoint, i.e. MSS_{cp}. MSS_{cp} responds with the checkpoint.

In order to retrieve the logs the current MSS sends requests to all the MSSs in the list *log_set*. Each MSS, upon receipt of the request responds with the log entries for the MH if it has any. The current MSS compiles the logs into a list ordered by time and sends it to the MH along with the checkpoint. In order to recover from the failure, the MH rolls back to this checkpoint and replays the logs in order. Once recovery is completed successfully, a checkpoint of the current state is taken and sent to the MSS and the local variables are reset.

4.3 Storage management at the mobile support stations

Even though movement based checkpointing helps to control the number of checkpoints depending on mobility and failure rates, the combination of checkpoints and logs for every MH can amount to a significant amount of information to be persisted at the MSSs. In [13], the authors explain that if storage at MSSs is depleted, they will either have to temporarily halt normal operation and perform garbage collection or find costly alternative storage for new checkpoints and logs. Since halting a MSS makes its local MHs inaccessible, it must be prevented.

Storage can be recovered by deleting outdated recovery information. Recovery information becomes outdated every time a new checkpoint is taken successfully which occurs during normal operation after every M handoffs and when a recovery operation executes successfully. Thus, when a MH takes a new checkpoint, the MSSs delete all recovery information related to the MH.

5 Performance analysis

The performance of the proposed scheme is evaluated using analytical methods by means of Stochastic Petri Net (SPN) modeling. A Petri net is a popular graphical and mathematical modeling tool used to describe and study concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic systems. We choose SPN because of its ability to deal with general time distributions for events, its concise representation of the underlying state machine to deal with a large number of states, and its expressiveness to reason about a MH's behavior as it migrates from one state to another in response to events occurring in the system.

Fig. 2 SPN model

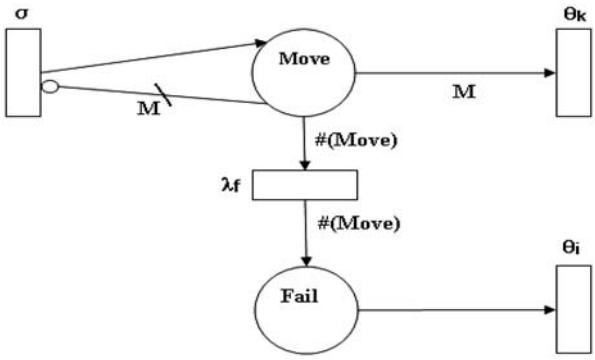

Table 1 SPN model parameters

Parameter	Description
σ	MH mobility rate, i.e. the rate at which the MH crosses cell boundaries
μ	Log arrival rate, i.e. the rate at which logs are created
λ_f	MH failure rate, i.e. the rate at which the MH fails
M	Movement threshold, i.e. the number of handoffs after which the MH takes a checkpoint
r	Ratio of bandwidth of wireless network to wired network
T_{ckp_w}	Time required to transmit a checkpoint through the wireless link
T_{log_w}	Time required to load a log entry through the wireless link
T_{eckp}	Time required to roll back to the last checkpoint
T_{elog}	Time required to execute a log entry at the MH
F_r	Probability of recovery
T_r	Recovery time
T	Recovery time deadline
θ_k	Checkpoint rate, i.e. $1/\theta_k$ is the time required to take a checkpoint
θ_i	Recovery rate, i.e. $1/\theta_i = T_r \,.\, i$ is the number of movements since the last checkpoint
N_{mss_logs}	Number of MSSs storing logs
D_{mss}	Average hop count between the MSS storing the checkpoint and the MSS in which the MH recovers

5.1 Model

The SPN model of a mobile computing system employing the movement-based recovery scheme is shown in Fig. 2.

The parameters controlling the model and their descriptions are given in Table 1.

Current literature [12] states that assuming an exponential distribution for the system parameters such as log arrival rate, failure rate and mobility rate, is a reasonable approach. Others have used models such as gamma, hyperexponential, lognormal, or hyper-Erlang distributions [4]. In this paper we take the former approach and use exponential distribution for these parameters which makes the underlying Markov model of the SPN tractable. As part of future work, we intend to analyze the system

behavior for other distributions and compare the results. The main characteristics of the SPN model are:

1. The model consists of 2 places and 4 transitions. Place "Move" represents the state in which the number of movements of the MH since the last checkpoint is not more than the threshold M and there has been no failure. Place "Fail" stands for the state in which a MH failure has occurred. Initially the MH is in a consistent state with a checkpoint in the current MSS and a zero value for the count of movement, as represented by the place "Move" without tokens.
2. The MH moves from one cell to another with a mobility rate σ. Whenever the MH encounters a handoff, the number of tokens in the place "Move" is increased by 1. This behavior is described by the transition on the left upper part of the SPN model. The MH stays in the state represented by place "Move" except in the event of a failure. When a failure occurs, this transition will not be enabled until failure recovery has completed. The only inhibitor arc ensures that the number of movements between two consecutive checkpoints is less than the threshold M. When the number of tokens in the place "Move" becomes equal to M, the upper right transition is fired. This transition requires time $1/\theta_k$ which is used to create a checkpoint of the current state and save it to the current base station. This transition also resets the handoff counter to 0 as represented by the M tokens on the output arc of the place "Move".
3. When a MH failure occurs, the system status migrates from the state "Move" to the state "Fail". All the tokens in the place "Move" move to the place "Fail". The recovery time $1/\theta_i$ depends on the number of movements since the last checkpoint which is denoted by $i = \#(\text{Fail})$, the number of tokens in the place "Fail".

The transition firing rates θ_k and θ_i require some elaboration:

- Parameter $\theta_k : \theta_k$ represents the execution rate to perform a checkpoint operation. During checkpointing, the MH takes a snapshot of its current state and sends it to its current MSS through the wireless channel. The MSS then stores it in its stable storage. Since the time taken for wireless transmission, T_{ckp_w} is significantly longer than the others, and the time spent on a checkpoint operation is approximated to this value. Accordingly $\theta_k = 1/T_{ckp\_w}$.
- Parameter $\theta_i : \theta_i$ represents the recovery rate of the MH and is the inverse of the recovery time, where i is the number of handoffs experienced by the MH since the last checkpoint and before failure. The recovery time includes (a) the time needed to send recovery information requests to the MSSs storing the latest checkpoint and all logs since the latest checkpoint, (b) the time required to transmit the latest checkpoint from the MSS where it is stored (MSS_{cp}) to the MSS in which the MH has recovered (MSS_{rec}) through the wired network and through the wireless channel to the MH, (c) the time required to transmit all the logs from the respective MSSs where they are located (MSS_{logs}) to the MSS_{rec} through the wired network and through the wireless channel to the MH and (d) the time required to rollback to the last checkpoint and apply all the logs at the MH.

Before describing the equation for computing the average recovery time, some variables are defined.

- N_{mss_logs}—This represents the number of MSSs storing logs. The exact value of this variable is the size of the list *log_set*. This value at most is the number of handoffs before failure, i.e. i, where every MSS through which the MH passed is unique and logs are created in every one of them. For simplicity, we assume that $N_{mss\_logs} = i$.
- $D_{mss,j}$—This is the average hop count between two MSSs separated by j handoffs. Thus for MSScp and MSSrec separated by i handoffs, $D_{mss,i}$ represents the hop count between MSScp and MSSrec. Based on the hexagonal network model, we calculate $D_{mss,j}$ as:

$$D_{mss,j} = 1 + (j-1)\left[\frac{1}{6} \times -1 + \frac{2}{6} \times 0 + \frac{3}{6} \times 1\right].$$

This equation shows that on the first move, the count increases by 1, but on each subsequent move the MH moves backward with probability 1/6 (hop count decreased by 1), sideways with probability 2/6 (hop count remained the same) and moves forward with probability 3/6 (hop count increased by 1). The equation above can then be reduced to $D_{mss,j} = (j+2)/3$.

Thus, the total time to recover after i movements is the sum of the following components:

- Time spent on recovery information requests as given by:

$$T_{rec\_req} = \sum_{j=1}^{i} (D_{mss,j} \times r \times T_{log\_w}).$$

Here we assume that the size of a request packet is not more than the size of a log entry packet. Hence for simplicity, we use the time to transmit a log entry T_{log_w} as the time to transmit a request packet. MSSrec needs to send a request message to each of the i MSSs including MSScp storing the checkpoint, which is i handoffs apart from MSSrec.
- Time spent on transmitting the latest checkpoint to the MH as given by:

$$T_{ckp\_tx} = D_{mss,i} \times r \times T_{ckp\_w} + T_{ckp\_w}.$$

The first part of this equation represents the time required to transmit the last checkpoint from MSS_{cp} to MSS_{rec}, assuming the distance between the two is $D_{mss,i}$.
- Time spent on transmitting the logs to the MH as given by:

$$T_{log\_tx} = \sum_{j=1}^{i} (D_{mss,j} \times n_{mss} \times r \times T_{log\_w}) + n \times T_{log\_w},$$

where n is the number of log entries accumulated since the last checkpoint and is given by $n = (\mu * i)/\sigma$ and n_{mss} is the number of log entries per MSS, given by $n_{mss} = (\mu * i)/[\sigma(i+1)]$. The denominator is $i+1$ because MSS_{cp} is the first MSS and MSS_{rec} is the last MSS storing log entries. $\sum_{j=1}^{i} (D_{mss,j} \times n_{mss} \times r \times T_{log\_w})$, the first term, accounts for the cost to transmit log entries from the i MSSs to

MSS_{rec} and the second term represents the time required from MSS_{rec} to transmit each log entry through the wireless channels.

- Time spent to rollback to the last checkpoint and apply the logs as given by: $T_{rec} = T_{eckp} + n \times T_{elog}$.

Hence, the total time to recover after i movements is given by:

$$T_r^i = T_{rec\_req} + \max(T_{ckp\_tx}, T_{log\_tx}) + T_{rec} \tag{1}$$

with the transition firing rate $\theta_i = 1/T_r^i$. The "max" operator is used to take the maximum of checkpoint transfer time and log transfer time since these messages can be transmitted to MSS_{rec} and subsequently to the MH concurrently.

The SPN model's underlying Markov model has $2M + 1$ states. If P_j denotes the probability of state j, the average recovery time per failure is given by:

$$T_r = \sum_{j=0}^{2M+1} P_j \times T_r^i. \tag{2}$$

The recovery probability F_r is defined as the probability that recovery time is less than or equal to recovery time deadline T and is given by:

$$F_r = \text{Prob}\{T_r \leq T\} = \sum_{S=1}^{2M+1} P_s \times (1 - e^{-\theta_S T}), \tag{3}$$

where θ_S corresponds to θ_i if the MH has made i handoffs in state S.

In addition, we define T_c as the total time spent on checkpointing and logging before a failure. It represents the total cost of recovery and is given as:

$$T_c = (\sigma/(M \times \lambda_f)) \times T_{ckp\_w} + (\mu/\lambda_f) \times T_{log\_w}, \tag{4}$$

where $\sigma/(M\lambda_f)$ denotes the total number of checkpoints before a failure, and (μ/λ_f) represents the total number of log entries before a failure. Thus, T_c represents the total failure-free operations cost and the second part of (4) depicts the total delay incurred by logging during failure free operations.

Equations (2) and (4) enable us to define the total cost of recovery per failure as the weighted sum of the average recovery time and the total time spent on the checkpointing and logging per failure. It is given by:

$$T_{cost} = w_1 T_c + w_2 T_r, \tag{5}$$

where $w_1 + w_2 = 1$, $w_1 > 0$, $w_2 > 0$ are the weights associated with recovery time and failure-free operation cost. Different applications have differing failure recovery performance requirements with respect to the time taken for recovery and checkpointing. The weighting of the components of the total cost of recovery allows the configuration of such performance requirements per application and user. We use $w_1 = w_2 = 0.5$ in the analysis below to account for the situation where T_{cost} is equally proportional to T_c and T_r.

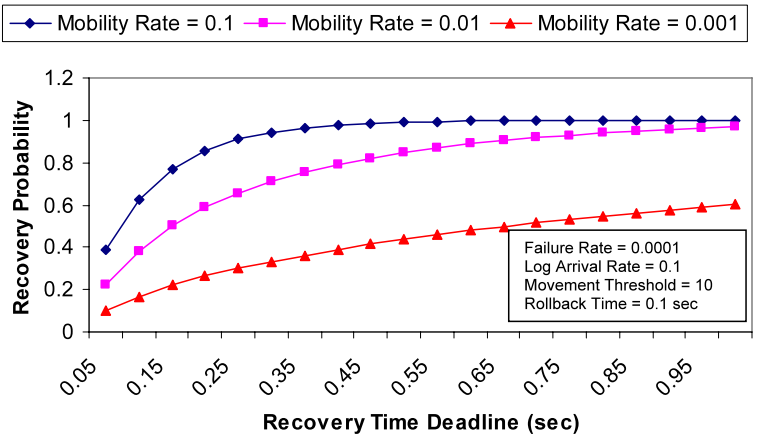

Fig. 3 Recovery probability vs. recovery time deadline

5.2 Results and analysis

The SPN model was implemented and analyzed using the SPNP software. The following parameter values were kept constant across all the runs. Specifically, the size of a log entry is 50 B, size of a checkpoint is 2 KB, bandwidth of the wired network is 2 Mbps, ratio of bandwidth of wireless to wired network (r) is 0.1, time to roll back to the previous checkpoint (T_{eckp}) is 0.1 sec, and time required to apply a log entry (T_{elog}) is 0.0001 s. Thus the time required to transmit a log entry through the wireless channel (T_{log_w}) is 0.002 s and the time required to transmit a checkpoint through the wireless channel (T_{ckp_w}) is 0.08 s. Model parameters such as recovery time deadline, mobility rate, log arrival rate, failure rate, and movement threshold were varied across runs. These values were chosen for analysis purposes only and do not assume a specific application or environment. The results presented here show system behavior for a wide range of parametric values which depicts the broad applicability of the scheme.

Figure 3 shows the probability of recovery against recovery time deadline for varying values for mobility rate. Recovery probability increases with increase in recovery time deadline. For a constant value of recovery time deadline, the probability increases with increase in mobility rate. This is because with higher mobility rate, the checkpoint interval reduces and the number of logs accumulated between checkpoints decreases. This enables faster recovery. From the curve for mobility rate 0.1, it can be seen that 90% of failures can be recovered in 0.3 seconds, and at most 0.5 seconds to achieve 100% recovery probability.

Figure 4 shows the effect of varying log arrival rates on the probability of recovery given a recovery time deadline of 0.3 sec. When the log arrival rate is low, the number of logs accumulated between the last checkpoint and failure is small and hence there is a high probability of recovery within 0.3 sec. However, as the log arrival rate increases, the percentage of failures that can be recovered within a fixed time decreases. Therefore, when employing the movement-based checkpoint strategy, it is necessary to estimate the rate of write events of the application in order to maximize efficiency

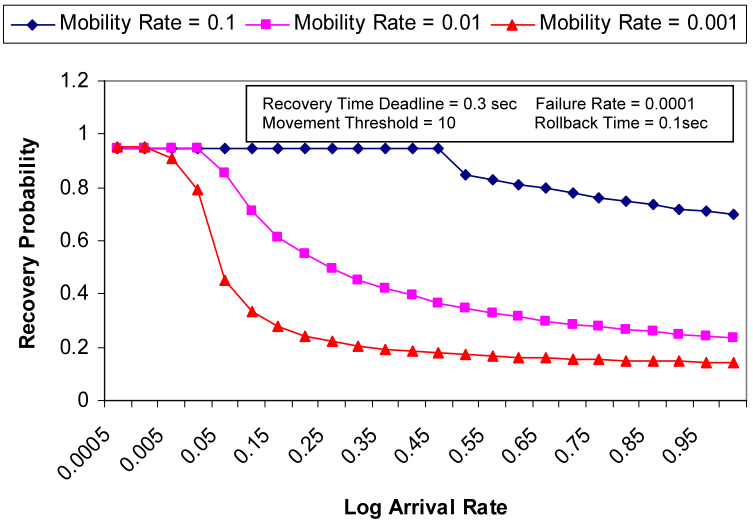

Fig. 4 Recovery probability vs. log arrival rate

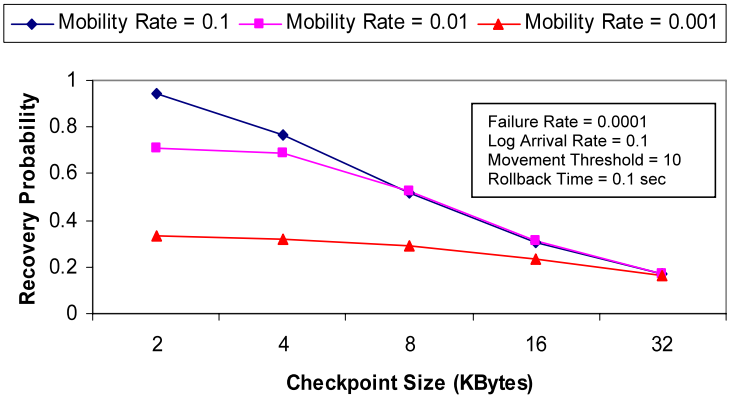

Fig. 5 Recovery probability vs. checkpoint size

of failure recovery. Also, observe that at low log arrival rates, the time required to transfer the checkpoint is greater than the time required to transfer logs. Thus, the checkpoint transfer time does not vary with the log arrival rate. This explains the flat segment of the curves. As the log arrival rate increases, the time required to transfer logs becomes greater than the checkpoint transfer time and the log arrival rate begins to have an effect on the recovery probability.

Figure 5 shows the effect of the size of a checkpoint on the probability of recovery, given a recovery time deadline of 0.3 sec. As the checkpoint size increases, the time required to transfer the checkpoint to MSS_{rec} increases and the probability of recovery decreases. Furthermore, when the checkpoint size is sufficiently large (greater than 32 KB in Fig. 5), the recovery probability becomes very low such that the result-

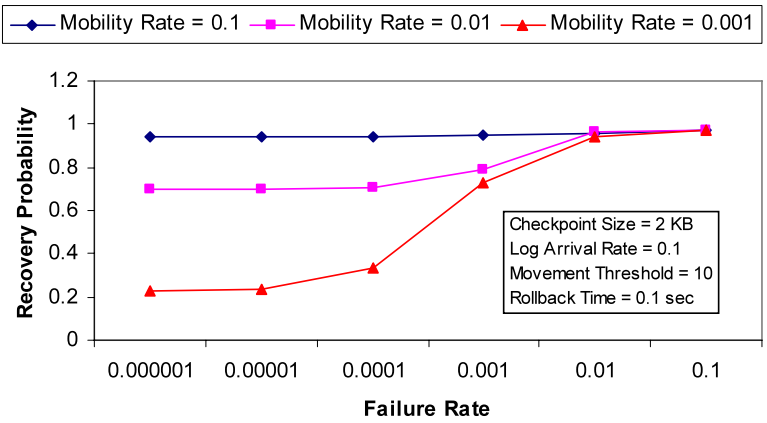

Fig. 6 Recovery probability vs. failure rate

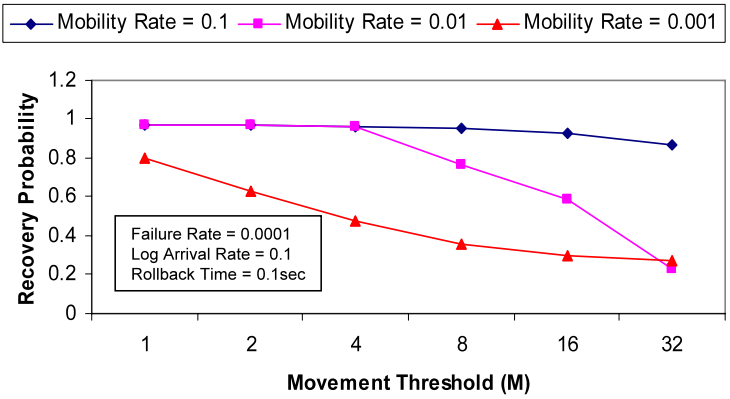

Fig. 7 Recovery probability vs. movement threshold

ing recovery probability is insensitive to the mobility rate because the time to transfer a large checkpoint dominates the time to transfer logs in recovery time.

Figure 6 shows the recovery probability for varying failure rates given a recovery time deadline of 0.3 sec. Failure rate affects the number of log entries accumulated between the last checkpoint and the current MH failure. The higher the failure rate, the fewer the log entries, and lesser the time required to recover. Therefore the recovery probability increases as the failure rate increases.

Figure 7 shows the effect of varying the movement threshold (the number of handoffs between two consecutive checkpoints) on the probability of recovery. Figure 8 shows the effect of varying the movement threshold on recovery time for the same parameters as above. The reason for the downward trend of the recovery probability curve is that for constant mobility and log arrival rates, when the movement threshold increases, the time interval between two checkpoints increases and more log entries would be created. Hence, the time spent on recovery increases and the probability of

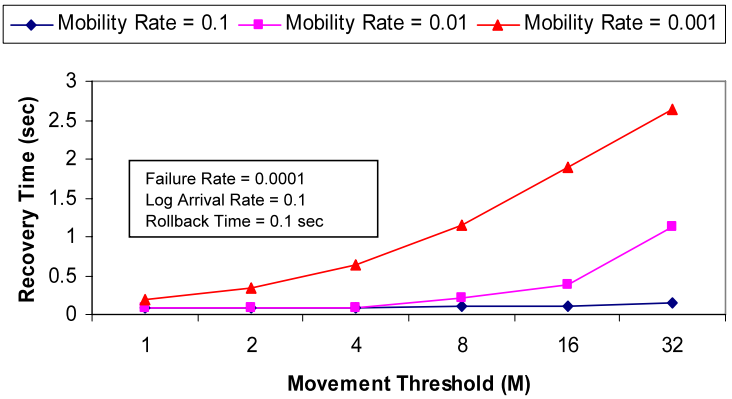

Fig. 8 Recovery time vs. movement threshold

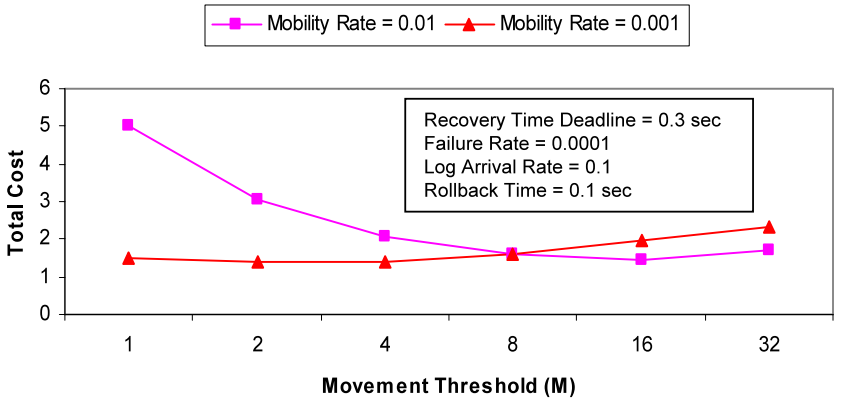

Fig. 9 Determining optimal movement threshold that minimizes recovery cost per failure

recovery in a given time decreases. However, it cannot be concluded that setting the movement threshold to 1 will produce the best results. Although doing so will decrease the recovery time greatly, the total number of checkpoints increases, resulting in significant additional overhead during failure-free operations for the creation and maintenance of checkpoints. Thus, there exists a tradeoff between recovery time and the total time spent on the checkpoints and logging for a fixed cost.

Next, we analyze the tradeoff between the recovery time (T_r) and the total time spent on the checkpointing and logging per failure (T_c) to identify the optimal movement threshold to minimize the total cost of recovery (T_{cost} as given by (5)). Figure 9 shows the relationship between the total recovery cost and the movement threshold M for varying values of mobility rate. The curves indicate that there exists an optimal movement threshold under a given operating condition. When $\lambda_f = 0.0001$, and $\sigma = 0.01$ the optimal value of M is 25 and when $\sigma = 0.001$ the optimal value of M is 3 (as shown in Fig. 9).

This optimal M value that minimizes T_{cost} is dictated by the operational conditions characterized by the MH's mobility rate and failure rate, as well as the mobile application's log rate. More specifically, when M is small, T_C dominates T_r because with a small threshold, checkpointing must be done frequently while recovery can be done quickly. As M increases, T_c decreases while T_r increases and the total cost drops. Finally when M is sufficiently large, T_r dominates T_C and the overall cost increases again. This tradeoff results in an optimal M value that minimizes the total cost. We also note that there is a cross-over M value beyond which lower mobility generates higher cost. For example, the cross-over point is $M = 8$ in Fig. 9. The reason is that when M is sufficiently high, T_r dominates T_C, and the increase in T_r is especially pronounced when the mobility rate is low. The reason that T_r is high when the mobility rate is low is that the checkpoint interval is longer (given the same M value) when the mobility rate is lower, thus resulting in more log entries being accumulated over the checkpoint interval and causing the system to take more time to load and execute log entries for failure recovery. The SPN model developed in the paper can easily identify the optimal M value to minimize the total recovery cost per failure, when given proper parameter values as input to the model.

6 Summary and applicability

In this paper we have presented an efficient failure recovery scheme for mobile computing systems based on movement-based checkpointing and logging. Current approaches take checkpoints periodically without regard to the mobility rate of the user and unnecessarily incur additional overhead in maintaining recovery data. Our movement-based checkpointing and logging scheme takes a checkpoint only after the mobile node has made M movements (mobility handoffs). The value of M is mainly governed by the failure rate, log arrival rate, and the mobility rate of the application and MH. A performance model has been developed based on stochastic Petri nets to identify the optimal movement threshold M, when given the failure, mobility and log arrival rates, to minimize the cost of recovery per failure, as well as to calculate the failure recoverability, when given an application specified recovery time deadline. The results of performance analysis show the sensitivity of recoverability to the various parameters.

To apply the results obtained in the paper, one can build a table at static time covering possible parameter values of the mobility rate and failure rate of the MH and log arrival rate of the mobile applications, and listing the optimal M value that would minimize the recovery cost per failure. Then at runtime based on the measured rates, the optimal M may be selected dynamically to minimize the recovery cost per failure. The optimal M selected must also satisfy the specified recovery probability when given an application deadline to recover from a failure.

As the next step, we plan to analyze and compare the proposed algorithm to existing approaches, especially the gain achieved over the use of constant periodic checkpointing. This paper assumed exponential distribution for the system parameters. A natural extension of this work is to study the nature of these parameters and their effect on system behavior. This work is based on mobile applications running

in wireless cellular networks. With the proliferation of Mobile IPv6 in future all-IP systems, we plan to extend the work to MIPv6 environments. Simple extensions to the proposed mobility-based checkpointing and logging algorithm exist and can be examined, such as buffering log entries at the MH and then periodically flushing accumulated log entries to the base station to further reduce the cost of logging during failure-free periods. We also plan to look at the implementation issue for realizing the movement-based checkpointing and logging scheme in mobile computing systems.

References

1. Acharya, A., Badrinath, B.R.: Checkpointing distributed applications on mobile computers. In: 3rd Int'l Con. on Parallel and Distributed Information Systems, pp. 73–80. October 1994
2. Chen, I.R., Gu, B., George, S., Cheng, S.T.: On failure recoverability of client-server applications in mobile wireless environments. IEEE Trans. Reliab. **54**(1), 115–122 (2005)
3. Elnozahy, E.N., Alvisi, L., Wang, Y.M., Johnson, D.B.: A survey of rollback-recovery protocols in message passing systems. Technical Report, School of Computer Science, Carnegie Mellon University (1996). CMU-CS-96-181
4. Fang, Y., Chlamtac, I., Lin, Y.-B.: Modeling PCS networks under general call holding time and cell residence time distributions. IEEE/ACM Trans. Netw. **5**(6), 893–906 (1998)
5. Gadiraju, S., Kumar, V.: Recovery in the mobile wireless environment using mobile agents. IEEE Trans. Mobile Comput. **3**(2), 180–191 (2004)
6. Higaki, H., Takizawa, M.: Checkpoint-recovery protocol for reliable mobile systems. In: 17th IEEE Symposium on Reliable Distributed Systems, pp. 93–99. October 1998
7. Neves, N., Fuchs, W.K.: Adaptive recovery for mobile environments. Commun. ACM **40**(1), 68–74 (1997)
8. Park, T., Yeom, H.Y.: Application controlled checkpointing coordination for fault-tolerant distributed computing systems. Parallel Comput. **26**(4), 467–482 (2000)
9. Park, T., Woo, N., Yeom, H.Y.: An efficient optimistic message logging scheme for recoverable mobile computing systems. IEEE Trans. Mobile Comput. **1**(4), 265–277 (2002)
10. Park, T., Woo, N., Yeom, H.Y.: An efficient recovery scheme for fault-tolerant mobile computing systems. Futur. Gener. Comput. Syst. **19**(1), 37–53 (2003)
11. Pradhan, D.K., Krishna, P., Vaidya, N.H.: Recoverable mobile environment: design and tradeoff analysis. In: Annual Symposium on Fault Tolerant Computing, pp. 16–25 (1996)
12. Wang, J., Zeng, Q.A., Agrawal, D.P.: Performance analysis of a preemptive and priority reservation handoff scheme for integrated service-based wireless mobile networks. IEEE Trans. Mobile Comput. **2**(1), 65–75 (2003)
13. Yao, B., Ssu, K.F., Fuchs, W.K.: Message logging in mobile computing. In: 29th Annual International Symposium on Fault-Tolerant Computing, pp. 294–301 (1999)