An Efficient Handoff Strategy for Mobile Computing Checkpoint System

Chaoguang Men[1,2], Zhenpeng Xu[2], and Dongsheng Wang[1,2]

[1] National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China
[2] Research Center of High Dependability Computing Technology, Harbin Engineering University, Harbin, Heilongjiang, 150001, P.R. China
{mencg,wds}@tsinghua.edu.cn,
{menchaoguang,xuzhenpeng}@hrbeu.edu.cn

Abstract. The *Eager*, *Lazy* and *Movement-based* strategies are used in mobile computing system when handoff. They result in performance loss while moving the whole checkpoint on fault-free or slow recovery while not moving any checkpoint until recovery. In the paper, a compromise strategy is proposed. The whole recovery information are broken into two parts, which one little part with high-priority should be transferred to the new cell during handoff and another large part with low-priority should be transferred only when the mobile host recovers from a fault. From the view of mobile host, it seems that all recovery information reside on the local mobile support station. The strategy guarantees little performance losing when fault-free and quick recovery when fault occurs. Experiments and analysis show the handoff strategy performance overcomes others.

Keywords: mobile computing, fault tolerant, checkpoint, handoff, rollback recovery.

1 Introduction

Checkpointing and rollback-recovery has been an attractive technique for providing fault-tolerance in mobile computing system [1]. Due to the mobility of the hosts, limited bandwidth, highly unreliable wireless link, mobile hosts disconnect from network voluntarily, power restriction and limitation of storage space in mobile devices, conventional checkpointing recovery schemes used in wired distributed network cannot be directly applied to mobile environment [2]. When a mobile host (*MH*) moves from one cell to another, *Eager*, *Lazy* and *Movement-based* strategies are used, which move the whole recovery information to new mobile support station (*MSS*) or not move any recovery information until a *MH* recovers [3]. A strategy which moves the whole recovery information in fault-free will depress the performance of system due to transfer useless information and others which not move any recovery information until a fault occurs will delay the system recovers from a fault due to recovery information can not be gotten in time. No one strategy is

T.-W. Kuo et al. (Eds.): EUC 2007, LNCS 4808, pp. 410–421, 2007.

excellent in every circumstance. A compromise strategy is proposed. Only a few part of recovery information is moved to the new local cell when a *MH* moves, which little useless work is done when fault-free or quick recovery can be done when a fault occurs.

The paper is organized as follows: Section 2 introduces the system model and definitions. Section 3 presents a checkpoint and recovery strategy with an efficient handoff scheme for mobile computing. Section 4 gives its correctness proofs. Section 5 compares the handoff scheme with others. Section 6 draws a conclusion.

2 Preliminaries

A mobile computing system $MCS=\langle N, C\rangle$ is composed of a set of nodes N and a set of channels C. The set of nodes $N=M\cup S$ can be divided into two types, $M=\{MH_1, MH_2,..., MH_n\}$ is the set of *MH*s, which are able to move while retaining their network connections and $S=\{MSS_1, MSS_2,..., MSS_m\}$ is the set of static nodes acting as the *MSS*s. The set of channels $C=W\cup W'$ can be divided into two disjoint sets, the set of high-speed wired channels W, where $W=S\times S$ is the type through that static nodes are connected, and the set of low bandwidth wireless channels W', where $W'=S\times M$ is the type through that *MH*s are connected to a *MSS*. A cell is a geographical area covered by a *MSS*. A *MH* residing in the cell of MSS_p can directly communicate with MSS_p through a wireless channel. In a cell of MSS_i, let $CL_i=\{MH_j \mid MH_j\in MSS_i, 0<j<n+1\}$ denotes the active nodes or sleeping nodes identified by $Active\_MH\_List_i$ or $Disconnected\_MH\_List_i$ respectively, then there exists a channel $\langle MSS_i, MH_j\rangle \in W'$ only if $MH_j\in CL_i \Rightarrow MH_j\notin CL_k$, $\forall k\neq i$, assuming that the geographical cells around each of the *MSS* do not overlap. The *MH*s have limited battery power and hence cannot keep communication with the *MSS*s for long, hence they often disconnect from the network. Such disconnections can be voluntary without any fault or involuntary due to abruptly running out of battery. The mobile computing system model is described in Fig. 1.

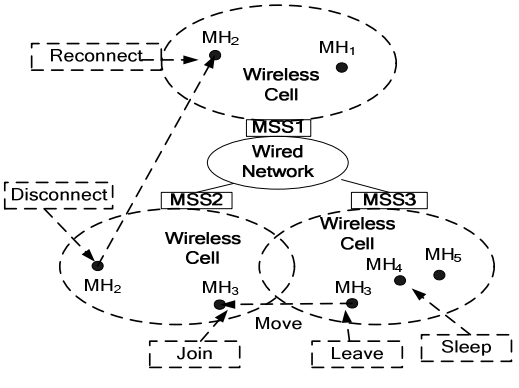

Fig. 1. Mobile computing system model

Distributed computations running concurrently on different *MH*s consist of a set of N processes denoted by P_1, P_2, ..., P_n. Processes do not share a global memory or a global physical clock, and they communicate with each other only through message passing. For simplicity, we assume that only one process runs on each *MH*. So we can use the terms '*MH*' and process interchangeably. We assume that each of the channels is bidirectional with reliable *FIFO* delivery of messages and the message transfer delays are finite but arbitrary. Processes follow the piece-wise deterministic execution model, and the underlying computation is asynchronous. The fault model is assumed to be *fail-stop* and all faults can be detected immediately, which results in halting failed process, initiating recovery action are considered to be transient and the same fault would not repeat when the process restarts.

Let $Rcv(i,\alpha)$ denotes the αth message receiving event of a process P_i; the state interval $I(i,\alpha)$ denotes the sequence of states generated between $Rcv(i,\alpha\text{-}1)$ and $Rcv(i,\alpha)$, where $\alpha>0$ and $Rcv(i,0)$ denotes the initial event. Then, the dependency relation of processes caused by the message communication can be defined as follows:

Definition 1. *Dependency Relation*: A state interval $I(i,\alpha)$ is said to be dependent on another state interval $I(j,\beta)$ if one of the following conditions is satisfied and the dependency relation is denoted by $I(j,\beta)\rightarrow I(i,\alpha)$:

(i). $i=j$ and $\alpha=\beta+1$
(ii). For an event $Rcv(i,\alpha)$, the corresponding message-sending event happens in $I(j,\beta)$
(iii). For any $I(k,\gamma)$, $I(j,\beta)\rightarrow I(k,\gamma)$ and $I(k,\gamma)\rightarrow I(i,\alpha)$ [2].

With the pessimistic message logging scheme, an interval $I(i,\alpha+1)$ can be fully recovered after a fault if the event, $Rcv(i,\alpha)$, has been stably logged; Otherwise, the interval becomes lost. During the rollback-recovery of a process, the dependency relation may cause an inconsistency problem.

Definition 2. *Orphan interval*: An interval on which depends any lost interval is called an orphan state interval.

Definition 3. *Consistent Recovery*: The recovery from a fault $F(i,f)$ is said to be consistent, if and only if there is not any orphan state interval, that is, for any $I(i,\alpha)\in L(i,f)$ there exists no $I(j,\beta)$, such that $I(i,\alpha)\rightarrow I(j,\beta)$. Where $F(i,f)$ denotes the fth fault of P_i and $L(i,f)$ denotes the set of lost state intervals caused by $F(i,f)$ [3].

The handoff and location scheme are supplied to support the mobility of *MH*. When a *MH* leaves a cell and enters another cell, it must end its current connection by sending a leave(r) message to its local *MSS*, where r is the sequence number of the last message received from the *MSS*. Then the *MH* establishes a new connection by sending a join(*MH*-id, previous *MSS*-id) message to the new *MSS*. Usually, leaving a cell and entering another cell happens simultaneously when an *MH* crosses the boundary between two cells and it is called a *handoff*. Each *MSS* maintains a list of identifiers of *MH*s that are currently supported by the *MSS*. A *MH* can also disconnect itself from the local *MSS* without leaving the cell by sending disconnect(r) message when the *MH* goes into the sleep mode for power conservation. Later, the *MH* can reconnect to any *MSS* by sending a reconnect(*MH*-id, previous *MSS*-id) message to

the *MSS*. If the *MH* is reconnected to a new *MSS*, the new *MSS* informs the previous *MSS* of the reconnection of the *MH* so that the previous *MSS* can perform the proper handoff procedures [4].

Handoff time is an important parameter which affects mobile system performance besides checkpoint state-saving cost and recovery cost [5]. There three categories handoff strategy named *Eager*, *Lazy* and *Movement-based*. *Eager* mobility handoff strategy, which also named *Pessimistic*, always keeps the logging and checkpoint information in the local *MSS* in which the *MH* currently resides [5]. Thus, when the *MH* moves from one *MSS* to another during the execution of a mobile application, all the checkpoint and logging information must be moved to the current *MSS* as well. The advantage of this approach is fast failure recovery. But the *MSS*s visited by the *MH* have to experience high fault-free cost to transfer the recovery information and access the stable storage. Under the *Lazy* strategy, on the other hand, the checkpoint and logging information do not be moved as the *MH* moves [5]. Rather, a forwarding pointer is established from the local *MSS* to the last *MSS* so that when a failure occurs, the checkpoint and logging information of the mobile application can be recovered from all the *MSS*s on the forwarding chain by following the links. The advantage of this approach is little fault-free cost, but the recovery cost can be too high, if the recovery information is dispersed over a wide range of cells. The tradeoff schemes are *Movement-based* handoff strategies, which are *Distance-based* and *Frequency-based* [4]. Under the *Distance-based* scheme, which focuses on the distance between MH_i and the *MSS* carrying latest checkpoint of MH_i, the checkpoint and message logs need to be moved into a *MSS* near MH_i, only when the moving distance of MH_i from a *MSS* carrying the latest checkpoint exceeds a certain threshold. On the other hand, the *Frequency-based* scheme concerns the number of handoffs, since that number indicates the number of sites carrying the message logs and the frequency of communication for collecting the message logs in case of recovery. Hence, in this scheme, MH_i keeps counting the number of handoff and transfers the checkpoint and logs if the number exceeds a certain value. Of course, in both of the above schemes, the recovery cost and the fault-free operation cost is adjustable using the threshold values. Checkpoint and logs are moved to new local *MSS* when fault-free, the *Movement-based* schemes have the disadvantage of *Eager*. Checkpoint and logs are not moved to new local *MSS* until recovery, the *Movement-based* schemes have the disadvantage of *Lazy*. Obviously, how effective these strategies would be depends on various system parameters, including the checkpoint rate, logging message arrival rate, user mobility rate, failure rate, and bandwidth. No one scheme is always better than others under all situations [6].

3 The Recovery Scheme

The proposed recovery scheme is based on independent checkpointing, pessimistic message logging and asynchronous rollback-recovery. An efficient handoff scheme is proposed. Different from *Eager*, *Lazy* and *Movement-based* schemes which move the whole recovery information or do not move any recovery information until recovery, in our strategy, the whole recovery information which include checkpoint and logs are broken into two parts. One part includes only a little part of recovery information with

high-priority. The other part includes the rest of the recovery information with low-priority. The high-priority part of recovery information is treated as that in *Eager* scheme and the low-priority part of recovery information is treated as that in *Lazy* scheme. Appropriate partitions high-priority and low-priority recovery information can satisfy both quick recovery and fault-free cost. When recovering from a fault, the high-priority part can be transferred instantly to the recovering *MH*, the low-priority part can be collected by the local *MSS* simultaneously from other *MSS*s and then be transferred to the recovering *MH* successively. From the view of the recovering *MH*, it seems that all recovery information always resides on the local *MSS*.

3.1 The Data Structure and Denotations

Let $CK_{i,\alpha}$ denotes the αth checkpoint of MH_i; CK_info_i is a record which contains six variables, *CK_sn*, *CK_loc*, *CK_low*, *Logm_seq*, *Send_max*, and *Log_queue*. *CK_sn* denotes the sequence number of the latest checkpoint and the *CK_loc* denotes the identifier of the *MSS* carrying the high-priority of the latest checkpoint; *CK_low* denotes the identifier of *MSS* carrying the low-priority of the latest checkpoint; *Logm_seq* denotes the sequence number of the first message logged after the latest checkpoint; *Send_max* denotes the maximum sequence number of message sent successfully by MH_i to other *MH*s since the latest checkpoint; *Log_queue* is a list established for the local *MSS* to save the identifiers of *MSS*s which have the logs saved after the latest checkpoint; $Msg_{i,\alpha}$ denotes the αth message sent by MH_i; Rcv_seq_i is an integer variable, which denotes the maximum sequence number of messages that have been received and consumed in MH_i. $Logm_{i,\alpha}$ denotes the αth message log.

3.2 The Checkpointing and Logging

Each MH_i takes an initial checkpoint on initialization and sets the corresponding checkpoint sequence number $CK\_info_i.CK\_sn$ to 0. Every *MH* takes checkpoint periodically. When MH_i finishes a new checkpoint, the information about this checkpoint is recorded in CK_info_i. The CK_info_i and the new checkpoint will be sent to its local MSS_p.

Each *MSS* logs the received messages before delivering to *MH*s in its cell. As a message heading for MH_i should be routed through the local MSS_p, using the local MSS_p to log the message into its storage space will not incur extra overhead. MSS_p also logs the messages of the mobility of *MH*s, including the messages of *MH*s to join in, leave from, disconnect from and reconnect to the cell. Upon a user input of the *MH*, a copy of it is firstly forwarded to the local MSS_p for logging in case of its lost. On receipt of the acknowledgment from MSS_p, the *MH* starts to process the input event. When MH_i leaves or disconnects from MSS_q, it sends *Disconnect*(*i*) message to the local MSS_q for logging. MSS_q logs the event on the receipt of it and deals with it. When MH_i joins in a new cell of *MSS*, says MSS_r, it sends *Join*(MSS_q) to MSS_r. And MSS_r will add MSS_p into the $CK\_info_i.Log\_queue$ if MSS_p is not in the $CK\_info_i.Log\_queue$.

The checkpointing and message logging algorithm is described in Fig. 2.

Actions taken when checkpointing Timer of MH_i Expires:
CK (i, ++$CK\_info_i.CK\_sn$);
/*saves the state of MH_i as a new checkpoint of MH_i.*/
$CK\_info_i.Logm\_seq=Rcv\_seq_i$+1;
/*assigns the sequence number of the first message which will be logged after the new checkpoint.*/
Send ($CK_{i,\alpha}$, CK_info_i, MSS_p) ;
/* Sends the new checkpoint and its information to its local MSS. */
Actions taken when MSS_p receives $CK_{i,\alpha}$ and CK_info_i from MH_i:
Store ($CK_{i,\alpha}$, CK_info_i);
/* Saves the new checkpoint and its information in local MSS_p. */
$CK\_info_i.CK\_loc= MSS_p$;
/* identifies the MSS carrying the high-priority of latest checkpoint.*/
$CK\_ low = MSS_p$;
/* identifies the MSS carrying the low-priority of latest checkpoint.*/
Actions taken when MH_i receives a message $Msg_{k,\alpha}$ from local MSS_p:
Consume $Msg_{k,\alpha}$;
/* MH_i deals with the message.*/
Rcv_seq_i ++;
/* MH_i adds the sequence number of message.*/
Actions taken when MSS_p receives a message $Msg_{i,\alpha}$ from MH_i:
Log $Msg_{i,\alpha}$;
$Logm_{i,\alpha}$++;
/* saves the new message into log-space and adds the message number. */
If $MSS_p \notin CK\_info_i.Log\_queue$
Then $CK\_info_i.Log\_queue = CK\_info_i.Log\_queue \bigcup \{MSS_p\}$;
/*add MSS_p to the $CK\_info_i.Log\_queue$. */
Transfer $Msg_{i,\alpha}$;
/* forwards the computational message to goal MH_i.*/
If ($Msg_{i,\alpha} \in$ {join, leave, disconnect, reconnect})
Actions ;
/* takes related actions according to the received messages.*/

Fig. 2. Checkpointing and message logging algorithm

3.3 The Handoff Strategy

The recovery information including checkpoint and massage logs are broken into two parts that one with high-priority and the other with low-priority. The main idea is that low-priority checkpoint information can be sent to the local MSS of recovering MH through high speed wired network at the same time as the high-priority recovery information is being sent to recovering MH through low speed wireless network.

The amount of high-priority part and low-priority part of recovery information depend on the speeds of wired and wireless networks. Set LT_{max} and WT_{min} denote the maximum communication speed of wireless network and minimum communication speed of wired network respectively. Set VP_0 denotes the amount of high-priority

recovery information. For simplicity, we assume that the amount transmitted is the integral multiple of packet size.

When a MH_i recovers from a fault, the VP_0 will be sent to MH_i from the local MSS_p instantly. The transmission time at least is:

$$t_1 = \frac{VP_0}{LT_{\max}} . \tag{1}$$

The amount of low-priority recovery information collected from other *MSS*s simultaneously in time t_1 at least is:

$$VP_1 = t_1 * WT_{\min} = \frac{VP_0 \times WT_{\min}}{LT_{\max}} . \tag{2}$$

Because the speed of wired network is faster than the speed of wireless, so $VP_1 > VP_0$, that is more information can be collected through wired network when an amount of information is sent through wireless network. In turn, more recovery information, VP_2, can be collected through wired network when VP_1 is sent through the wireless network. The amount of recovery information transferred to MSS_p from other *MSS*s is:

$$VP_0 \times \left[\frac{WT_{\min}}{LT_{\max}} + \left(\frac{WT_{\min}}{LT_{\max}} \right)^2 + \cdots + \left(\frac{WT_{\min}}{LT_{\max}} \right)^{n-1} + \left(\frac{WT_{\min}}{LT_{\max}} \right)^n \right] . \tag{3}$$

The effective handoff scheme is described in Fig. 3.

Actions taken when MSS_p receives join (MSS_q) request:
Send (VP_0_*retrieve*, MSS_q)
/*sends request message to MSS_q get high-priority part of $CK_{i,\alpha}$ ***/**
Actions taken when MSS_q receives VP_0_*retrieve* from the new local MSS_p:
Send (VP_0 , MSS_p)
/*sends high-priority part of recovery information to MSS_p. */
Actions taken when MSS_p receives VP_0:
CK_loc=MSS_p;
Store *CK_info*$_i$ and high-priority part of $CK_{i,\alpha}$;
/* identifies the *MSS* carrying the high-priority of latest checkpoint and saves the high-priority of recovery information.*/

Fig. 3. An effective handoff scheme

3.4 Independent Recovery

When recovering from a fault, MH_i sends a recovery request, *RollbackReq*(*i*), to its local MSS_p. If *CK_info*$_i$ is saved in the local MSS_p, the high-priority part of recovery information is sent to MH_i instantly. To collect the rest of checkpoint and logs, the local MSS_p sends *Chkpt_retrieve*(*CK_info*$_i$.*CK_sn*) and *Log_retrieve*(*CK_info*$_i$.*Logm_seq*) request to other *MSS*s according to *CK_low* and *CK_info*$_i$.*Log_queue*. After receiving the request, the other *MSS*s reply with the low-priority checkpoint part and

the logs whose sequence number is not less than $CK\_info_i.Logm\_seq$. After transferring the high-priority part of recovery information to the recovering MH_i, the local MSS_p sends the low-priority checkpoint part to the recovering MH_i successively. After receiving the whole checkpoint, MH_i reloads the checkpoint to restore the system, and then resumes and replays the logs. During recovering, new message sent to MH_i is saved in its local MSS_p, and will be forwarded to MH_i, in turn, after recovering. The message that sequence number less than $CK\_info_i.Logm\_seq$ is discarded to avoid repeat messages.

If there is not CK_info_i in the local MSS_p, it means that a fault has occurred in other cell and then MH_i enters this cell in where it submits the recovery request. MSS_p broadcasts recovering request to all *MSS*s. The previous local MSS_q sends the high-priority part of recovery information to MSS_p. MSS_p executes the recovering process. The asynchronous recovery algorithm is described in Fig. 4.

Actions taken when MH_i occurs a fault:
Send (*RollbackReq*(i), MSS_p);
/*Sends recovery request to the local MSS_p.*/
Actions taken when MSS_p receives *RollbackReq*(i) from MH_i:
If ($MH_i \in$ (*Active_MH_List*$_p$ *or Disconnected_MH_List*$_p$))
Send (VP_0, MH_i);
/*if MSS_p holds latest checkpoint with high-priority, sends it to MH_i. */
Send (*Chkpt_retrieve* (*CK_info*$_i$.*CK_sn*), *CK_low*);
/* sends retrieval message to *CK_low* to get the remnant part of checkpoint.*/
Send (*Log_retrieve*(*CK_info*$_i$.*Logm_seq*), *CK_info*$_i$.*Log_queue*);
/* sends request to MSS_r in the list of *Cpinfo*$_i$.*Log_queue* to get message logs.*/
Else
Broadcast *info_retrieve*(i);
/* If the local MSS_p don't hold the latest CK_info_i, broadcasts the recovery request.*/
Actions taken when MSS_k receives Broadcast *info_retrieve*(i) from MSS_p:
Send (high-priority of $CK_{i,\alpha}$, CK_info_i , MSS_p);
/* sends the high-priority part of recovery information to local MSS_p.*/
Actions taken when MSS_q receives *Chkpt_retrieve*(*CK_sn*) from MSS_p:
Send (the remnant of $CK_{i,\alpha}$, MSS_p);
/*sends the remnant of checkpoint to local MSS_p.*/
Actions taken when MSS_q receives *Log_retrieve* (*Logm_seq*) from MSS_p:
If ($\exists Logm_{i,\alpha}$ in MSS_q log space and $\alpha \geq Logm\_seq$)
Send ($Logm_{i,\alpha}$, MSS_p);
/*sends related message logs to local MSS_p.*/
Actions taken when MH_i receives the full $CK_{i,\alpha}$ from MSS_p:
Restore $CK_{i,\alpha}$;
/*restores the full checkpoint.*/
Resume action;
/*starts the computation process of recovery.*/

Fig. 4. The asynchronous recovery algorithm

3.5 Garbage Collection

MH_i takes new checkpoint $CK_{i,\alpha}$ and sends the checkpoint and CK_info_i to MSS_p. MSS_p sends a message which a new checkpoint has been taken to all *MSS*s which will delete old checkpoint $CK_{i,\alpha\text{-}1}$ and relative information that denoted by old $CK\_info_i.CK\_loc$, $CK\_info_i.CK\_low$, $CK\_info_i.Log\_queue$. Every *MSS* will release the space held by checkpoint $CK_{i,\alpha\text{-}1}$ and implement garbage collection.

4 Correctness of the Algorithm

Theorem 1**.** The asynchronous recovery of MH_i is a consistent recovery.

Proof: *Recoverable*: the latest checkpoint and the messages with the sequence number larger than the $CK\_info_i.logm\_seq$ were saved safely due to the reliable communication, the reliable *MSS*s and the pessimistic message logging. Therefore, on the recovery of MH_i, every message logs and the latest checkpoint can be retrieved. The messages can be replayed according to the sequence number after restoring the latest checkpoint. In other words, MH_i can reconstruct one possible sequence of state intervals as those constructed before the fault due to processes following the piece-wise deterministic execution model. So MH_i is recoverable on fault in the strategy.

Consistent recovery: The lost events which incurs $L(i,f)$ can only be the messages or user inputs that had not been sent successfully to the local *MSS* before a fault. This implies the corresponding messages could not be transferred to their destinations. According to the *definition* 1, the lost events can not incur new dependency relations between *MH*s. Therefore, for any $I(i,\alpha) \in L(i,f)$ there exists no $I(j,\beta)$, such that $I(i,\alpha) \rightarrow I(j,\beta)$. The independent recovery is a consistent recovery as the recovery strategy satisfies the *definition* 3. □

5 Performance Study

The model and parameters in [6] are adopted. *MH*s communicate with *MSS*s through 802.11a wireless network adapter. *MH* moves from one cell to another follows a Poisson process with rate σ=0.01. The message sending rate of a *MH* follows a Poisson process with rate λ=0.1. Each *MH* takes a checkpoint with a fixed interval Tc=1000s, the failure rate of each *MH* follows an exponential distribution with rate δ=0.0001. Increment strategy is adopted for saving a checkpoint and its size is 1MB. The size of a logs entry is 50B. The ratio of wireless network speed to wired network speed is r=0.1. The time required to load a log entry through a wireless channel is T_1=0.016s, and the time required to load a checkpoint through a wireless channel is T_3=0.32s. The time required to execute a log entry is T_2=0.0008s. We assume that when a *MH* moves 5 times, its checkpoint should be moved to the new local *MSS* in *Frequency-based* strategy and when a *MH* moves 10 times, its checkpoint should be moved to the new local *MSS* in *Distance-based* strategy.

Fig. 5 shows the amounts of recovery information needed to be transmitted in every recovery strategies. The *y*-axis indicates the overhead of message transfer incurred by different strategies for *MH*'s recovery, while the *x*-axis denotes the time that a fault

occurs on the *MH*. The overhead of recovery information management under our virtual strategy is always less than those under *Eager*, *Frequency-based* and *Distance-based* strategy, and only a little large than that under *Lazy* strategy, because our compromise strategy only moves little, not the whole, latest checkpoint to the local *MSS* when a *MH* moves from one cell to another.

Fig. 6 shows the amounts of recovery information needed be transmitted by every strategy under different message sending rate after the system has run 500 seconds. The overhead of every handoff strategy becomes increment with the increment of message sending rateλ. The overhead of our handoff strategy almost equals to that under *Lazy* and far less than the other's.

Let $N(t)$ denotes the number of logs saved until *MH* faults. $f_f(t)$ denotes the probability when a fault occurs in time t. Under our and *Eager* strategies, the recovering probability of time T is [6]:

$$F_1(T) = \sum_{M=0}^{+\infty} \int_{MTc}^{(M+1)Tc} \Pr\{ N(t)[T_1 + T_2] + T_3 \le T \} f_f(t) dt \cdot \tag{4}$$

M denotes the checkpoint number experienced by a *MH* in time t, T_1, T_2 and T_3 denote the mean time of loading a log entry through wireless network, executing a log entry and loading whole checkpoint through wireless network respectively. $N(t)$ is a Poisson process with rate λ=0.1, and $f_f(t)$ is an exponential distribution with rate δ=0.0001, we get:

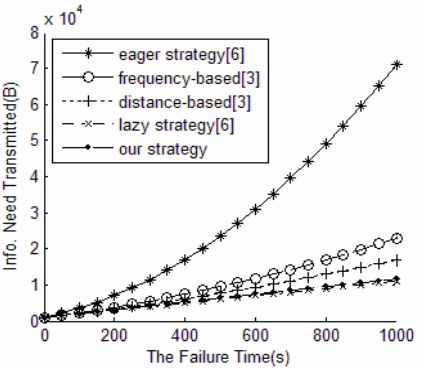

Fig. 5. The amounts of recovery information need transmitted

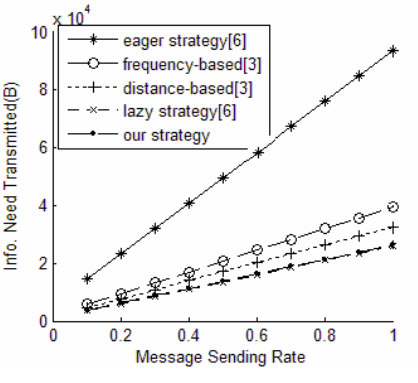

Fig. 6. The amounts of recovery information need transmitted with different rate

$$F_1(T) = \sum_{M=0}^{+\infty} \int_0^{Tc} \sum_{n=0}^{\left\lfloor \frac{T-T_3}{T_1+T_2} \right\rfloor} \frac{e^{-\lambda t'} (\lambda t')^n}{n!} \bullet \delta e^{-\delta t'} \bullet e^{-\delta MTc} dt'$$

$$= \frac{\int_0^{Tc} \sum_{n=0}^{\left\lfloor \frac{T-T_3}{T_1+T_2} \right\rfloor} \frac{e^{-\lambda t} (\lambda t)^n}{n!} \bullet \delta e^{-\delta t} dt}{1 - e^{-\delta Tc}} . \tag{5}$$

Under *Lazy* strategy, the recovering probability of time T is:

$$F_2(T) = \sum_{M=0}^{+\infty} \int_{MTc}^{(M+1)Tc} \Pr\{T_r \le T \mid t_f = t, k_b = k\} \bullet \Pr\{k_b = k \mid t_f = t\} f_f(t) dt \,. \tag{6}$$

Replacement $N(t)$ and $f_f(t)$ by their value:

$$F_2(T) = \sum_{M=0}^{+\infty} \int_0^{Tc} \sum_{k=0}^{+\infty} \sum_{n=0}^{\left\lfloor \frac{T-T_3-rT_3-rkT_3}{rT_1+T_1+T_2} \right\rfloor} \frac{e^{-\lambda t'}(\lambda t')^n}{n!} \bullet \frac{e^{-\sigma t'(\sigma t')^k}}{k!} \bullet \delta e^{-\delta t'} \bullet e^{-\delta MTc} dt'$$

$$= \frac{\int_0^{Tc} \sum_{k=0}^{+\infty} \sum_{n=0}^{\left\lfloor \frac{T-T_3-rT_3-rkT_3}{rT_1+T_1+T_2} \right\rfloor} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \bullet \frac{e^{-\sigma t(\sigma t)^k}}{k!} \bullet \delta e^{-\delta t} dt}{1 - e^{-\delta Tc}} \,. \tag{7}$$

Fig. 7 shows the probabilities of recovering time under various handoff strategies. Our strategy which only has a little overhead large than that under *Eager* is better than that under *Lazy* and has the same recovery probability as *Eager* has. Fig. 8 shows the executing overhead under our asynchronous recovery strategy and coordinated recovery strategy which the number of *MH*s is 100 and only 10 *MH*s need recovery from a fault. As shown in the figure, our strategy is better than coordinated strategy, and is more effective.

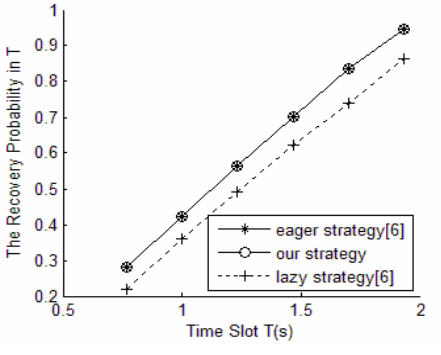

Fig. 7. The fault recovery probability

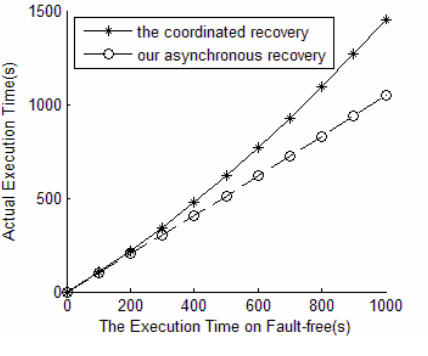

Fig. 8. The actual execution time of the mobile applications

6 Conclusion

The handoff strategy taken when a *MH* moves from one cell to another will affect the executing efficiency and recovering time of checkpoint algorithm. Different from other schemes, in the strategy proposed in the paper, the recovery information is

broken into two parts, which the first part must be transferred instantly to the new cell when a handoff happens and the second part can be transferred simultaneously to the local *MSS* through static network as the first part is transferred to the recovering *MH*. The partition principle is that the first part as little as possible and the second part as large as possible under guaranteeing recovering information to be transmitted to recovering *MH* successively. From the view of recovering *MH*, it seems that all recovery information resides on the local *MSS* all the time. This strategy considers both minimum executing time on fault-free and quickly recovering from a fault. Experiments and analysis show our strategy is better than others.

References

1. Elnozahy, E.N., Alvisi, L., Wang, Y.M., Johnson, D.B.: A Survey of Rollback-Recovery Protocols in Message-Passing Systems. ACM Computing Surveys 34(3), 375–408 (2002)
2. Ching, E.Y., Phipatanasuphorn, V.: A Survey of Checkpoint-Recovery Techniques in Wireless Networks (2002), http://www.cae.wisc.edu/ ece753/papers/Paper_9.pdf
3. Park, T., Woo, N., Yeom, H.Y.: An Efficient Optimistic Message Logging Scheme for Recoverable Mobile Computing Systems. IEEE Transactions on Mobile Computing 1(4), 265–277 (2002)
4. Park, T., Woo, N., Yeom, H.Y.: An Efficient Recovery Scheme for Mobile Computing Environments. In: The 8th International Conference on Parallel and Distributed Systems (ICPADS), Kyongju City, Korea, pp. 53–60 (2001)
5. Pradhan, D.K., Krishna, P., Vaiday, N.H.: Recoverable Mobile Environment: Design and Trade-off Analysis. In: Proc. of the 26th Int'l Symp. on Fault Tolerant Computing System, Sendai, Japan, pp. 16–25 (1996)
6. Chen, I.-R., Gu, B., George, S.E., Cheng, S.-T.: On failure recoverability of client-server applications in mobile wireless environments. IEEE Transactions on Reliability 54(1), 115–122 (2005)