Checkpoint and Recovery for Reliable Mobile Computing

Hiroaki Higaki and Makoto Takizawa
Department of Computers and Systems Engineering
Tokyo Denki University
{hig,taki}@takilab.k.dendai.ac.jp

Abstract

Information systems consist of multiple mobile stations and fixed stations communicating with each other. Mission critical applications are required to be executed fault-tolerantly in these systems. However, mobile stations support neither enough volume of storage and processing power nor enough capacity of battery to do reliable communication for a long period. Moreover, wireless communication channels are less reliable. Hence, the communication channels with the mobile stations are often disconnected. Therefore, it is difficult for multiple mobile stations to take checkpoints synchronously since the communication channels with the mobile stations may be disconnected even during taking the checkpoints. We have proposed hybrid checkpointing where checkpoints are taken asynchronously by the mobile stations and synchronously by the fixed stations. In addition, the mobile stations record messages for getting local states consistent with the checkpoints taken by the fixed stations. In this paper, we popose the method how the mobile stations record the messages, gather the messages stored in the stable storages distributed in multiple mobile stations, and recompute the messages in the consistent order.

1 Introduction

According to the advances of communication and computer technologies, many kinds of mobile stations like notebook computers and personal data assistants (PDAs) are widely available. Intelligent Transport Systems (ITSs) with mobile communications are now being developed. New computation paradigms like *nomadic* computing [3] are also proposed.

A mobile system is composed of *fixed stations* and *mobile stations* interconnected by communication networks. The fixed stations are located at the fixed locations in the network. The mobile stations move from one location to another in the network. The network is divided into a number of *cells*, i.e. mobile stations move from one cell to another. There is a *mobile support station* (MSS) in each cell. A mobile station communicates with another station only through the MSS. The MSSs and the fixed stations are interconnected by the high-speed network. The network addresses of the mobile stations are automatically assigned by using DHCP (Dynamic Host Configuration Protocol) [7]. The connections with the mobile stations can be automatically maintained by the mobile protocols [14, 18, 19] even if the mobile stations move among the cells. The mobile stations sometimes move out of the cells and do not have so much capacity of battery that the communication with the other stations can be continued for a long period. Hence, the communication channels with the mobile stations may be disconnected. However, some applications are computed on mobile and fixed stations and are required to be continued even while the communication channel is disconnected. Many papers [4, 9, 11] discuss how to handle the *disconnected operations*.

The *checkpoint-restart* [5,6,10,12,17,20–22] is one of the well-known methods to realize reliable distributed systems. Every station s_i takes a checkpoint c_i where the local state information of s_i is stored in the stable storage. If some station fails, s_i restarts the computation from c_i. A set of checkpoints taken by all the stations is required to be *consistent* [6]. A fixed station F_i can easily take checkpoints consistent with the others by using *synchronous* distributed checkpointing protocols [6,8,12,17,20] since F_i can communicate with each other by using the high-speed network and have enough volume of stable storage to store the state information. Papers [13, 16] discuss how the mobile stations M_i take the checkpoint c_{M_i} *synchronously* in the stable storage. However, it is difficult for M_i to take c_{M_i} due to the lack of stable storage and battery capacity. Moreover, it gets more difficult for F_i and M_i to take checkpoints synchronously if the communication channels between M_i and the MSSs are often disconnected.

We assume that every MSS S_i is equipped with enough volume of stable storage to store the local state information of all the mobile stations in the cell of S_i. M_i takes c_{M_i} by storing the local state information in S_i. M_i may fail to take c_{M_i} due to the lack of battery capacity or the movement to the outside of the cell. If the checkpoints are taken synchronously, all the stations have to give up to take the checkpoints if some mobile station fails to take the checkpoint. Hence, *asynchronous* checkpointing protocols [5,10,21,22] are preferable for the mobile stations. Papers [1, 15] propose the mobile *asynchronous* checkpointing protocols. Here, the protocol overhead is high since S_i is required to take a new checkpoint of M_i each time a message is transmitted between them. In this paper, we newly propose a *hybrid checkpointing* protocol where the checkpoints are asynchronously taken by the mobile stations while synchronously by the fixed stations. Here, a checkpoint c_{M_i} of M_i is taken only when M_i sends a checkpointing request to S_i. Hence, the number of accesses to the stable storages of the MSSs can be reduced. Therefore, the hybrid checkpointing protocol makes the mobile systems so reliable that the mission critical applications can be computed

with less overhead.

The rest of this paper is organized as follows. In section 2, we show the system model. In section 3, we overview the hybrid checkpointing protocol. In section 4, the recovery protocol for the hybrid checkpointing protocol is discussed.

2 System Model

A distributed system $\mathcal{S} = \langle \mathcal{V}, \mathcal{L} \rangle$ is composed of multiple stations $\mathcal{V} = \{s_1, \ldots, s_n\}$ interconnected by communication channels $\mathcal{L} \subseteq \mathcal{V}^2$. The computation is realized by cooperation of multiple stations communicating with each other by exchanging messages through the channels. $\langle s_i, s_j \rangle \in \mathcal{L}$ indicates a channel from s_i to s_j. We assume that each channel $\langle s_i, s_j \rangle$ is reliable and bidirectional. In s_i, two kinds of events occur: communication events and local events. A local state of s_i is assumed to be changed when a communication event, i.e. a message-sending event $s(m)$ or a message-receipt event $r(m)$ of a message m, occurs. Hence, a local state of s_i is determined by the initial state and the sequence of communication events occurring in s_i.

In a mobile computing system $\mathcal{MS}$, there are three kinds of stations: *fixed stations* $F_1, \ldots, F_f$, *mobile stations* $M_1, \ldots, M_m$ and *mobile support stations* (MSSs) $S_1, \ldots, S_s$ as shown in Figure 1. Every F_i is connected at the fixed location of the network. Each M_i moves from one location to another. If M_i is in a cell supported by S_j, M_i communicates with S_j by using the wireless or cable communication channel. S_j forwards messages from M_i to the destination stations and delivers the messages from the other stations to M_i. The connection between M_i and another station is automatically maintained by the cooperation of the MSSs even if M_i moves among the cells [14, 18, 19]. The fixed stations and the MSSs are interconnected by the high-speed network.

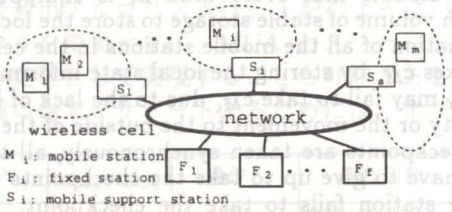

Figure 1: Mobile computing system.

Each M_i does not have so much capacity of battery that M_i can continue to communicate with an S_j for a long period. Hence, M_i often disconnects the connection with another station in order to reduce the power consumption of the battery while the applications are being computed in M_i. Furthermore, since M_i has neither enough computation power nor enough volume of storage like hard-disks, it is difficult for M_i to take checkpoints by itself. In this paper, we discuss a way where a mobile station stores the local state information in a stable storage of an MSS at a checkpoint and messages sent and received by the mobile station after the checkpoint are also stored in the MSS to realize consistent recovery.

3 Hybrid Checkpointing

3.1 Overview

The computation in $\mathcal{MS}$ is realized by cooperation of mobile stations $M_1, \ldots, M_m$ and fixed stations $F_1, \ldots, F_f$. Each M_i is in one of the cells supported by MSSs $S_1, \ldots, S_s$. Here, M_i is *supported by* S_j and S_j is a *current* MSS of M_i. The stations exchange messages by using the mobile communication protocol [14, 18, 19]. That is, each stations can communicate with the others without being conscious of the locations of the stations.

The advantage of the synchronous checkpointing protocols is that the computation can be restarted without domino effect. However, it is difficult for M_i to take c_{M_i} synchronously. Here, we propose a *hybrid checkpointing protocol* which has the following properties:

- The fixed stations take local checkpoints by using the synchronous checkpointing protocol. A collection of the checkpoints taken by the fixed stations is referred to as a *coordinated* checkpoint.
- The mobile stations take local checkpoints by using the asynchronous checkpointing protocol.

The state information of M_i at c_{M_i} is stored in the stable storage of S_j which is the current MSS of M_i. In addition, the messages sent and received by M_i are also stored in the stable storage of S_j. M_i fails to take c_{M_i} if the channel between M_i and S_j is disconnected owing that M_i moves out of the cell or the battery of M_i is exhausted during taking c_{M_i}. Thus, M_i takes c_{M_i} when M_i surely could take c_{M_i}. That is, M_i takes c_{M_i} only if M_i does not move out of the cell and has enough capacity of battery to take c_{M_i}. Therefore, M_i asynchronously takes c_{M_i}, i.e. independently of the other stations.

3.2 Checkpointing protocol

In the hybrid checkpointing protocol, the fixed stations $F_1, \ldots, F_f$ synchronously take a coordinated checkpoint $CC = \langle c_{F_1}, \ldots, c_{F_f} \rangle$ while the mobile stations $M_1, \ldots, M_m$ asynchronously take local checkpoints $c_{M_1}, \ldots, c_{M_m}$. Each M_i has to restart the computation from a state consistent with CC. However, c_{M_i} is not always consistent with CC because M_i takes c_{M_i} independently of the other stations. Hence, M_i restarts the computation by using the *message log* [2]. Here, the messages sent and received after c_{M_i} by M_i are stored in the message log in the stable storage of the current MSS S_j. If M_i restarts the computation, M_i recomputes the messages stored in the message log to get the state consistent with CC.

Suppose M_i is supported by S_i^j. Since every message sent and received by M_i is transmitted via S_i^j, the message can be stored in the stable storage of S_i^j even if M_i has no stable storage. A *checkpoint agent* process A_i^j in S_i^j records messages sent and received by M_i in the message log ml_i^j on behalf of M_i. Moreover, A_i^j takes the local checkpoint c_{M_i} of M_i by recording the state information in the state log sl_i^j if M_i requests A_i^j to take c_{M_i}.

$F_1, \ldots, F_f$ take CC by using the following protocol proposed in [12]:

[Coordinated Checkpoint CC]

1) A *coordinator* station CS sends a request message $Creq$ to $F_1, \ldots, F_f$ and $S_1, \ldots, S_s$.
2) On receipt of $Creq$, each F_i and S_j take a tentative checkpoint tc_{F_i} and tc_{S_j}, respectively, and send back a reply message $Crep$ to CS.
3) If CS receives $Creps$ from all the stations, CS sends a final message $Cfin$ to $F_1, \ldots, F_f$ and $S_1, \ldots, S_s$.
4) On receipt of $Cfin$, each F_i and S_j makes tc_{F_i} and tc_{S_j} permanent, i.e. c_{F_i} and c_{S_j}, respectively.

In order for CC to be consistent, each station suspends the computation and the transmission of application messages while the station has a tentative checkpoint.

Next, we discuss how M_i takes c_{M_i}. Here, suppose M_i is supported by S_i^j. The checkpoint agent A_i^j in S_i^j takes a *tentative* local checkpoint tc_{M_i} independently of the other stations. The state information required for M_i to restart the computation from tc_{M_i} is carried by a tentative checkpoint request message $TCreq$. On receipt of $TCreq$, A_i^j stores the state information of M_i in the *tentative state log* tsl_i^j in the volatile storage of S_i^j.

[Tentative checkpoint tc_{M_i} in A_i^j]

1) M_i sends $TCreq$ to A_i^j. $TCreq$ carries the state information SI_i of M_i.
2) On receipt of $TCreq$, A_i^j takes tc_{M_i} of M_i by storing SI_i in tsl_i^j. If some checkpoint agent A_i^k $(k < j)$ has taken another tentative checkpoint tc'_{M_i} of M_i, A_i^j requests A_i^k to discard tc'_{M_i}.

Let $\langle A_i^1, \ldots, A_i^c \rangle$ be a sequence of checkpoint agents where A_i^1 has tc_{M_i} and A_i^c is the current checkpoint agent of M_i. If S_i^c receives $Creq$, A_i^1 makes tc_{M_i} a permanent checkpoint c_{M_i} by moving the state information from tsl_i^1 in the volatile storage to sl_i^1 in the stable storage of S_i^1. In addition, A_i^k $(1 \leq k \leq c)$ moves the messages from tml_i^k in the volatile storage to ml_i^k in the stable storage of S_i^k.

[Permanent checkpoint c_{M_i} in A_i^j]

- If S_i^1 receives $Creq$, A_i^1 moves the state information from tsl_i^1 to sl_i^1 before S_i^1 sends back $Crep$.
- If S_i^k $(k \neq 1)$ receives $Creq$, A_i^k moves the messages from tml_i^k to ml_i^k before S_i^k sends back $Crep$.

If there is another permanent checkpoint c'_{M_i} when c_{M_i} is taken, c'_{M_i} and the messages for the recovery can be discarded from the stable storage after taking c_{M_i}.

There are three cases with respect to in which order A_i^j receives $Creq$ and $TCreq$ messages:

1) If A_i^j receives $TCreq$ before $Creq$, i.e. A_i^j takes tc_{M_i} before receipt of $Creq$, tc_{M_i} is changed to c_{M_i}. That is, the messages in tml_i^j and the state information in tsl_i^j in the volatile storage are stored in ml_i^j and sl_i^j in the stable storage, respectively [Figure 2].
2) If A_i^j receives $TCreq$ and $TCreq'$ successively, i.e. A_i^j takes tc_{M_i} on receipt of $TCreq$ and receives $TCreq'$ without receiving $Creq$, tc_{M_i} is discarded and A_i^j takes another tentative checkpoint tc'_{M_i}. The messages in tml_i^j recorded between tc_{M_i} and tc'_{M_i} are discarded [Figure 3].
3) If A_i^j receives $Creq$ and $Creq'$ successively, i.e. A_i^j takes c_{M_i} on receipt of $Creq$ and receives $Creq'$ without receiving $TCreq$, c_{M_i} is still a permanent checkpoint. The messages in tml_i^j are stored in ml_i^j [Figure 4].

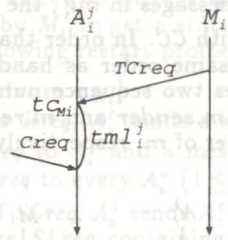

Figure 2: $TCreq$ and $Creq$ (1).

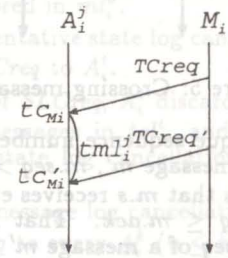

Figure 3: $TCreq$ and $Creq$ (2).

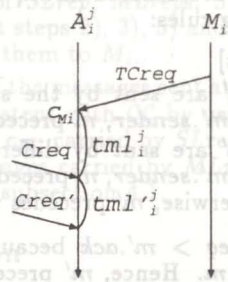

Figure 4: $TCreq$ and $Creq$ (3).

The hybrid checkpointing protocol has the following properties:

- Each M_i has one permanent checkpoint c_{M_i} consistent with the most recent CC.
- Each M_i has at most one tentative checkpoint tc_{M_i}.

4 Recovery Protocol

4.1 Message ordering for recovery

Suppose A_i^j receives two messages m and m' destined to M_i in this order. M_i receives m and m' forwarded by A_i^j in the same order. Next, suppose M_i sends m and m' in this order. A_i^j forwards m and m' to the destinations in the same order as M_i sends. Then, suppose M_i sends m' after receipt of m. A_i^j forwards m to M_i before receipt of m'. In these cases, A_i^j can keep the sequence of the messages exchanged with M_i in ml_i^j. However, if M_i sends m' before receipt of m, A_i^j may receive m' after sending m as shown in Figure 5. This means that A_i^j cannot know the occurrence sequence of the communication events in M_i. Hence, if M_i restarts the computation from c_{M_i} and recomputes the messages in ml_i^j, the state of M_i may be inconsistent with CC. In order that A_i^j records the messages in the same order as handled in M_i, each message m carries two sequence numbers $m.seq$ and $m.ack$. Here, let $m.sender$ and $m.receiver$ mean the sender and receiver of m, respectively.

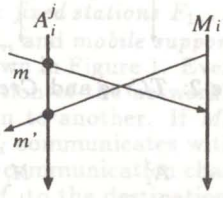

Figure 5: Crossing messages.

- m has a unique sequence number $m.seq$. If m is sent after a message m', $m.seq > m'.seq$.
- $m.ack$ means that $m.s$ receives every message m' where $m'.seq \leq m.ack$. That is, $m.ack$ is the same as $m'.seq$ of a message m' that is the most recently received message by $m.sender$.

If A_i^j sends or receives messages m and m' in this order, m and m' are ordered in ml_i^j according to the following ordering rules:

[Ordering rules]

- If m and m' are sent by the same sender, i.e. $m.sender = m'.sender$, m precedes m'.
- If m and m' are sent by different senders, i.e. $m.sender \neq m'.sender$, m precedes m' if $m.seq \leq m'.ack$. Otherwise, m' precedes m.

In Figref 5, $m.seq > m'.ack$ because M_i sends m' before receipt of m. Hence, m' precedes m although A_i^j sends m before m'. Thus, A_i^j stores m' before m in ml_i^j and a sequence of messages in ml_i^j is the same as M_i handles the messages.

Suppose that M_i is initially supported by S_i^1 and moves from S_i^k to S_i^{k+1} $(1 \leq k < c)$. S_i^c is the current MSS of M_i. In each S_i^k, there exists a checkpoint agent A_i^k of M_i. $\langle A_i^1, \ldots, A_i^c \rangle$ is a sequence of checkpoint agents and A_i^c is a *current* checkpoint agent of M_i. Each A_i^k stores the messages exchanged with M_i in a *tentative message log* tml_i^k in the volatile storage of S_i^k. Hence, a sequence of messages that M_i has sent and received are stored in a sequence of the message logs $\langle tml_i^1, \ldots, tml_i^c \rangle$.

4.2 Message logging for recovery

In $\langle A_i^1, \ldots, A_i^c \rangle$, suppose A_i^1 and A_i^t $(1 < t \leq c)$ have c_{M_i} and tc_{M_i}, respectively. That is, A_i^1 and A_i^t receive $TCreq$ from M_i and some A_i^u $(1 < u < t)$ receives $Creq$. Since A_i^v $(1 \leq v \leq u)$ stores the messages exchanged with M_i in ml_i^v, M_i gets a state consistent with CC by computing the messages in ml_i^v from c_{M_i} at which the state information is in sl_i^1. The messages forwarded by A_i^k $(u < k \leq c)$ are stored in tml_i^k. When tc_{M_i} taken by A_i^t is changed to c_{M_i} on receipt of $Creq$, some messages in tml_i^k $(u < k < t)$ can be discarded since these messages never be recomputed for restarting the computation of M_i from c_{M_i}. Here, we discuss which messages have to be stored in the stable storage of A_i^j.

Suppose A_i^j sends a messages m to M_i while receiving $TCreq$ and $Creq$. There are the following four cases:

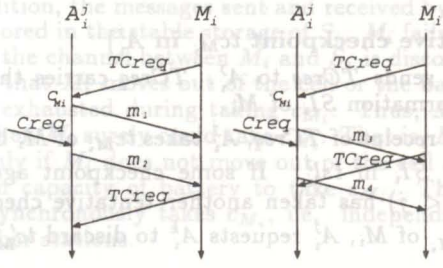

Figure 6: Logging m from A_i^j to M_i.

1) A_i^j sends m after receipt of $TCreq$ and before receipt of $Creq$ like m_1 in Figure 6. M_i recomputes m if M_i is restarted from c_{M_i}. Hence, m is stored in ml_i^j in the stable storage on receipt of $Creq$.
2) A_i^j sends m after receipt of $Creq$ and before receipt of $TCreq$, and M_i receives m before sending $TCreq$ like m_2 in Figure 6. Since M_i restarts the computation from a state consistent with CC without m, m is discarded.
3) A_i^j sends m before receipt of $Creq$, and M_i receives m after sending $TCreq$ like m_3 in Figure 6. M_i recomputes m if M_i is restarted from c_{M_i}. Hence, m is stored in ml_i^j in the stable storage on receipt of $Creq$. In addition, m has to be recorded for M_i to restart the computation from tc_{M_i} if tc_{M_i} is changed to be permanent. Thus, m is still in tml_{ij} in the volatile storage even after receipt of $TCreq$.
4) A_i^j sends m after receipt of $Creq$ and before receipt of $TCreq$, and M_i receives m after sending $TCreq$ like m_4 in Figure 6. Though M_i restarts the computation from a state consistent with CC without m, m has to be recorded for M_i to restart the computation from tc_{M_i} if tc_{M_i} is changed to

be permanent. Thus, m is recorded in tml_i^j even after receipt of *TCreq*.

A messages which can be discarded is referred to as *insignificant*. When A_i^j forwards m to M_i, A_i^j cannot identify which case from 1) to 4) m shows. Thus, A_i^j records every message in tml_i^j in the volatile storage of S_i^j. If A_i^j receives *TCreq* and m is insignificant, A_i^j discards m from tml_i^j.

Next, suppose A_i^j receives a messages m from M_i while receiving *TCreq* and *Creq*. There are following two cases:

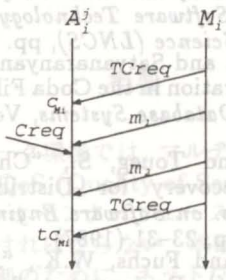

Figure 7: Logging m from M_i to A_i^j.

1) A_i^j receives m after receipt of *TCreq* and before receipt of *Creq* like m_1 in Figure 7. M_i recomputes m for restarting from c_{M_i}. Hence, A_i^j stores m in ml_i^j in the stable storage on receipt of *Creq*.
2) A_i^j receives m after receipt of *Creq* and before receipt of *TCreq* like m_2 in Figure 7. If $m.ack < m_l.seq$ where m_l is the message most recently sent by A_i^j to M_i before receipt of *Creq*, M_i recomputes m for restarting from c_{M_i}. Hence, A_i^j stores m in ml_i^j in the stable storage on receipt of *Creq*. Otherwise, m is discarded.

The procedure for logging the messages in A_i^j is as follows:

[Message logging in A_i^j]

- On sending m to M_i, m is stored in tml_i^j.
- On receipt of m from M_i, m is stored in tml_i^j if some A_i^k $(k < j)$ has tc_{M_i}. If no A_i^k has tc_{M_i} and $m.ack < m_l.seq$ where m_l is the message most recently transmitted from A_i^j to M_i before receipt of the most recent *Creq*, m is stored in ml_i^j. Otherwise, m is discarded.
- On receipt of *TCreq*, $m \in tml_i^j$ transmitted from A_i^j to M_i is removed from tml_{ij} and discarded if $m.seq \geq TCreq.ack$.
- On receipt of *Creq*, all the messages in tml_i^j are stored in ml_i^j.

4.3 Restart protocol for recovery

We discuss how to restart the fixed stations and the mobile stations if some station is faulty. $F_1, \ldots, F_f$ restart the computation from CC by using the restart protocol in [12].

[Restarting F_i from $c_{F_i} \in CC$]

1) A *coordinator* station CS sends a request messages *Rreq* to $F_1, \ldots, F_f$ and $S_1, \ldots, S_s$.
2) On receipt of *Rreq*, each F_i and S_j send back a reply message *Rrep* to CS.
3) If CS receives *Rreps* from all the stations, CS sends a final messages *Rfin* to $F_1, \ldots, F_f$ and $S_1, \ldots, S_s$.
4) On receipt of *Rfin*, each F_i and S_j restart the computation from c_{F_i} and c_{S_j}, respectively.

In order to restart $M_1, \ldots, M_m$ from states consistent with CC, the mobile agents have to cooperate. Let $\langle A_i^1, \ldots, A_i^c \rangle$ be a sequence of checkpoint agents of M_i where A_i^1 has c_{M_i}, some A_i^t $(1 < t \leq c)$ has tc_{M_i}, and A_i^c is the current agent. That is, A_i^1 and A_i^t receive *TCreq* and some A_i^u $(1 \leq u \leq t)$ receives *Creq*. The messages transmitted between M_i and A_i^v $(1 \leq v \leq u)$ are stored in ml_i^v in the stable storage and recomputed by M_i to get a state consistent with CC. Here, the following restart protocol is used:

[Restarting M_i from c_{M_i}]

1) If S_i^c receives *Rreq*, A_i^c sends a state log request message *SLreq* to A_i^1 and a message log request message *MLreq* to every A_i^v $(1 \leq v \leq u)$.
2) On receipt of *SLreq*, A_i^1 sends A_i^c back a state log reply message *SLrep* containing the state information at c_i stored in sl_i^1.
3) On receipt of *MLreq*, each A_i^v sends A_i^c back a message log reply message *MLrep* containing the messages stored in ml_i^v.
4) A_i^c sends a tentative state log cancellation request message *SLCreq* to A_i^t.
5) On receipt of *SLCreq*, A_i^t discards tc_{M_i}, i.e. discards the messages in tsl_i^t, and sends A_i^c back a tentative state log cancellation reply message *SLCrep*.
6) A_i^c sends a message log cancellation request message *MLCreq* to every A_i^k $(u \leq k < c)$.
7) On receipt of *MLCreq*, A_i^k discards the messages in tml_i^k and sends A_i^c back a message log cancellation reply message *MLCrep*.
8) On receipt of *SLrep*, *MLreps*, *SLCrep* and *MLCreps* sent at steps 2), 3), 5) and 7), respectively, A_i^c forwards them to M_i.
9) On receipt of the messages sent at step 8), M_i gets a state consistent with CC by using the state information at c_{M_i} carried by *SLrep* and recomputing the messages carried by *MLreps* in the order discussed in subsection 4.1.

5 Evaluation

We evaluate the following checkpointing protocols in terms of the total processing time:

- Synchronous checkpointing protocol : every mobile station synchronously takes the checkpoint.
- Hybrid checkpointing protocol : every mobile station asynchronously takes the checkpoint.

Suppose there are n mobile stations $M_1, \ldots, M_n$. It is assumed to take L [sec] for each M_i to take the checkpoint. Here, we assume $L = 60$ and no message

transmission delay between the stations.

In the synchronous checkpointing protocol, the checkpoint c_i of M_i is taken only if all the stations successfully take the checkpoints. If M_i fails, all the other stations have to throw away the effort to take the checkpoints and the stations have to restart the checkpointing procedure again. Let f be a probability that M_i fails to take the checkpoint, which is computed to be 0.12 as presented in section 3. M_i takes 60[sec] to take the checkpoint by sending the state information of M_i to the current MSS. The probability that at least one mobile station fails during the checkpointing procedure is given $1-(1-f)^n$. The expected total processing time ET_S to take the checkpoints is $\frac{nL}{2}(1-f)^n(2+3(1-(1-f)^n)+4(1-(1-f)^n)^2+\ldots) = \frac{nL}{2}(1+\frac{1}{(1-f)^n})$.

In the hybrid checkpointing protocol, each mobile station M_i asynchronously takes the checkpoint. If M_i fails to take the checkpoint, M_i restarts the checkpointing procedure from the beginning. Even if M_i fails, the other stations do not have to restart the checkpointing procedure. The expected processing time for M_i to take the checkpoint is $\frac{L}{2}(1-f)(2+3f+4f^2+\ldots)$. Hence, the expected total processing time ET_H is $\frac{nL}{2}(1+\frac{1}{(1-f)})$.

6 Concluding Remarks

It is significant to discuss how to make the mobile systems more reliable and available. In order to realize the reliable mobile computation, we have discussed how to take the checkpoints and restart the computation in the mobile stations and the fixed ones. In this paper, we have proposed the recovery protocol for the *hybrid checkpointing protocol* where the mobile stations asynchronously take the local checkpoints and the fixed ones synchronously take the local checkpoints. We will evaluate the proposed protocols in a simulation and an implementation of a prototype system.

References

[1] Acharya, A. and Badrinath, B.R., "Checkpointing Distributed Applications on Mobile Computers," *Proc. of the 3rd International Conference on Parallel and Distributed Information Systems*, pp. 73–80 (1994).

[2] Alvisi, L., Hoppe, B., and Marzullo, K., "Nonblocking and Orphan-Free Message Logging Protocols," *Proc. of the 23rd International Symposium on Fault-Tolerant Computing*, pp. 145–154 (1993).

[3] Bagrodiu, R., Chu, W.W., Klienrock, L., and Popel, G., "Visim, Issues, and Architecture for Nomadic Computing," *IEEE Personal Communication*, Vol. 2, No. 6 (1985).

[4] Barbara, D. and Imielinski, T., "Sleepers and Workaholics: Caching Strategies in Mobile Environments," *Proc. of ACM SIGMOD*, pp. 1–12 (1994).

[5] Bhargava, B. and Lian, S.R., "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems," *Proc. of the 7th International Symposium on Reliable Distributed Systems*, pp. 3–12 (1988).

[6] Chandy, K.M. and Lamport L., "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Trans. on Computer Systems*, Vol. 3, No. 1, pp. 63–75 (1985).

[7] Dromos, R., "Dynamic Host Configuration Protocol," *RFC 1541* (1993).

[8] Higaki, H., Sima, K., Tanaka, K., Tachikawa, T. and Takizawa, M., "Checkpoint and Rollback in Asynchronous Distributed Systems," *Proc. of the 16th IEEE INFOCOM*, pp. 1000–1007 (1997).

[9] Huang, Y., Sistla, P., and Wolfson, O., "Data Replication for Mobile Computers," *Proc. of ACM SIGMOD*, pp. 13–24 (1994).

[10] Juang, T.T.Y. and Venkatesan, S., "Efficient Algorithms for Crash Recovery in Distributed Systems," *Proc. of the 10th Conference on Foundations of Software Technology and Theoretical Computer Science (LNCS)*, pp. 349–361 (1990).

[11] Kistler, J.J. and Satyanaranyanan, M., "Disconnected Operation in the Coda File System," *ACM Trans. on Database Systems*, Vol. 10, No. 1, pp. 2–25 (1992).

[12] Koo, R. and Toueg, S., "Checkpointing and Rollback-Recovery for Distributed Systems," *IEEE Trans. on Software Engineering*, Vol. SE–13, No. 1, pp. 23–31 (1987).

[13] Neves, N. and Fuchs, W.K., "Adaptive Recovery for Mobile Environments," *Communications of the ACM*, Vol. 40, No. 1, pp. 69–74 (1997).

[14] Perkins, C., "IP Mobility Support," *Internet Draft:draft-ietf-mobileip-protocol-12.txt* (1995).

[15] Pradhan, D.K., Krishna, P.P. and Vaidya, N.H., "Recovery in Mobile Wireless Environment: Design and Trade-off Analysis," *Proc. of the 26th International Symposium on Fault-Tolerant Computing*, pp. 16–25 (1996).

[16] Prakash, R. and Singhal, M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 10, pp. 1035–1048 (1996).

[17] Randell, B., "System Structure for Software Fault Tolerance," *IEEE Trans. on Software Engineering*, Vol. SE–1, No. 2, pp. 220–232 (1975).

[18] Tanaka, R. and Tsukamoto, M., "A CLNP-based Protocol for Mobile End Systems within an Area," *Proc. of IEEE ICNP-93*, pp. 64–71 (1993).

[19] Teraoka, F., Uehara, K., Sunahara, H., and Murai, J., "VIP: A Protocol Providing Host Mobility," *Comm. ACM*, Vol. 37, No. 8, pp. 67–75 (1994).

[20] Tong, Z., Kain, R.Y., and Tsai, W.T., "Rollback Recovery in Distributed Systems Using Loosely Synchronized Clocks," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 3, No. 2, pp. 246–251 (1992).

[21] Venkatesh, K., Radhakrishnan, T., and Li, H.F., "Optimal Checkpointing and Local Recording for Domino-Free Rollback Recovery," *Information Processing Letters*, Vol. 25, pp. 295–303 (1987).

[22] Wood, W. G., "A Decentralized Recovery Protocol," *Proc. of the 11th International Symposium on Fault Tolerant Computing Systems*, pp. 159–164 (1981).