**The Chinese University of Hong Kong**

**Department of Computer Science and Engineering**

Ph.D. – Term Paper

| | |
|---|---|
| Title: | Fault Tolerance and Performance Analysis in Wireless CORBA |
| Name: | Chen Xinyu |
| Student I.D.: | 01409620 |
| Contact Tel. No.: | 2609-8427 |
| Email A/C: | xychen |
| Supervisor: | Prof. Michael R. Lyu |
| Markers: | Prof. Jerome Yen (SEEM) & Prof. John C.S. Lui |
| Mode of Study: | Full-time |
| Submission Date: | November 28, 2002 |
| Term: | 2 |
| Fields: | |
| Presentation Date: | December 9, 2002 (Monday) |
| Time: | 10:30 am – 11:30 am |
| Venue: | Rm. 1021, Ho Sin-Hang Engineering Building |

## Abstract

*The emerging mobile wireless environment poses exciting challenges for distributed fault tolerant (FT) computing. This term paper first proposes a message logging and recovery protocol on the top of Telecom Wireless CORBA and FT-CORBA architectures. It uses the storage available at the Access Bridge as the stable storage to log messages and checkpoints on behalf of a mobile host. Our approach engages both quasi-sender-based and receiver-based logging methods and makes seamless handoff in the presence of faults. The details of how to tolerate mobile host disconnection, mobile host crash and Access Bridge crash are described. Then we analyzes the execution performance and the average availability of equi-number checkpointing with given message number.We will show how checkpointing and handoff influence these two metrics, under what conditions checkpointing is beneficial, and the optimal checkpointing interval for minimizing the total program execution time and maximizing the average availability.*

## Contents

## 1. Introduction

Advances in wireless networking technology and portable information appliances have brought about a new paradigm of decentralized computing, called mobile computing [8]. A mobile computing system is considered as an extension of distributed systems, in which much of the action takes place at the middleware level. CORBA (Common Object Request Broker Architecture), standardized by Object Management Group (OMG), is one of the most popular middlewares for distributed systems nowadays. OMG also published Telecom Wireless CORBA specification [17] to provide wireless access and terminal mobility in CORBA.

Mobile computing enables users to access and exchange information while they roam around in mobile environments. But it causes physical damage of mobile hosts (MHs) more probable [15]. MHs inherit slow processors and small memories. The wireless link usually suffers with a high bit error rate, a little bandwidth, and a long transfer delay. Mobile terminals even disconnect from the hosting network intermittently [23]. Mobile systems are more often subject to environmental conditions which can cause loss of communications or data [10]. All of these cause transient failures more frequent. Thus, mobile computing requires techniques to provide fault tolerance to continue its services despite such permanent and transient failures.

OMG has specified Fault Tolerant CORBA (FT-CORBA) [18] as a standard. FT-CORBA is based on entity replication. It employs three replication styles: Cold Passive, Warm Passive and Active Replication. Logging and checkpointing mechanisms record messages and entity states in logs. All these are intended for wired networks. How to provide fault tolerance in a mobile computing environment is a new challenge for decentralized systems. This term paper proposes a message logging and recovery protocol on the top of Wireless CORBA and FT-CORBA architectures [2]. The storage available at the Access Bridge is employed as the stable storage to log messages and checkpoints on behalf of an MH. Both the quasi-sender-based and the receiver-based logging methods are engaged in our approach . The checkpointing strategy is equi-number checkpointing. The Access Bridge hides mobile host disconnection from other network hosts [26] and makes a seamless handoff when fault tolerant properties are called upon. We also discuss how to tolerate mobile host disconnection, mobile host crash and Access Bridge crash.

After we engage the checkpointing, message logging and rollback techniques in mobile environments, how would the performance be changed and what benefit would we get? Checkpointing avoids the failed program to rollback to its beginning. However, the benefit of checkpointing comes at a price. Excessive checkpointing would result in performance degradation, while deficient checkpointing would still incur an expensive recovery overhead [11]. Therefore, a trade-off exists. There are numerous mathematical models have been proposed to evaluate the execution performance and to derive the optimal checkpointing interval. But these models assume that the

total program execution time with no failure is known in advance. In mobile environments, applications are distributed by sending and receiving computational and control messages. The total execution time is dependent on multiple factors, such as link bandwidth etc., so it varies during different runs. But the number of total computational message received is not changed. Another special factor that should be considered is handoff. In this term paper, we will use these observations to analyze the execution performance and the average availability with and without equi-number checkpointing strategy. We will derive the expected program execution time and the average availability under some assumptions. We will show how checkpointing and handoff influence these two metrics, under what conditions checkpointing is beneficial, and the optimal checkpointing interval for minimizing the total program execution time and maximizing the average availability. At last we will show the results of simulations and the comparisons between the performance and availability of programs with and without checkpointing.

## 2. Wireless CORBA Architecture

Before delving into the details of how to provide fault tolerance and performance and availability analysis, let's take a look at Wireless CORBA architecture briefly. Figure 1 shows the architecture, which identifies three

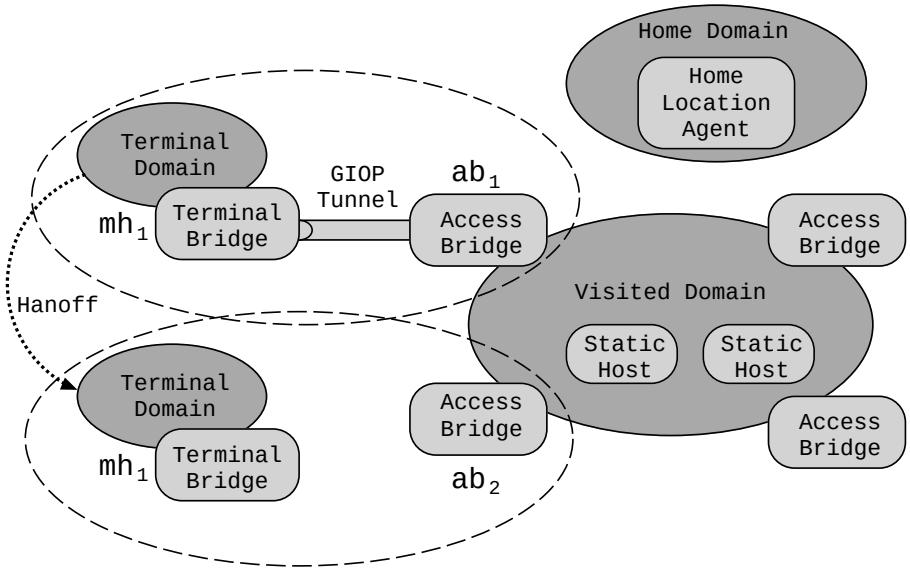



**Figure 1. Wireless CORBA architecture**

different domains: Terminal Domain, Visited Domain and Home Domain [17]. The Terminal Domain is an MH which can move around while maintaining network connections by a wireless interface. The MH hosts a Terminal Bridge (TB) through which the objects on the MH can communicate with objects in other wired or wireless networks. The Visited Domain contains several Access Bridges (ABs) to provide communication with

some objects on MHs. It also contains some Static Hosts (SHs). All communications in the Visited Domain are via wired links. ABs reside on MSSs which have necessary wireless facilities to communicate with MHs and have wired interfaces to communicate with static hosts and other MSSs. The Home Domain is composed by the Home Location Agent (HLA) which keeps track of the ABs that the MH has associated with when it moves around.

Each MSS has a geographical area within which it can communicate with MHs directly, which is plotted as dashed circle in Figure 1. When an MH moves across the border of the geographical area, a *handoff* occurs between the new AB and the old AB.

All hosts communicate with each other by messages only. The GIOP (General Inter-ORB Protocol) tunnel is the communication channel, through which the GTP (GIOP Tunnel Protocol) messages are transmitted between an AB and a TB. The GTP messages can be classified into two categories: control message and computational message. No messages can be exchanged among TBs directly. All messages to and from an MH are relayed by its currently associated ABs. During handoff, no computational messages can be transmitted between the ABs and the MH.

## 3. Fault Tolerant Wireless CORBA

Figure 2 presents an architectural overview of our FT Wireless CORBA. Our approach is based on checkpointing and message logging, which are the well-known techniques to minimize loss of computation in the presence of failures by periodically save the programs' states and log transmitted messages on stable storage during failure free execution. Each of the saved states is called a *checkpoint* [5], and the saving process of such states is called *checkpointing*. After a failure, there is a *repair* process which brings the failed device back to normal operation. Following the repair, the process of reloading the program status saved at the most recent checkpoint is often called a *rollback*. The reprocessing of the program, starting from the most recent checkpoint, applying the logged messages and until the point just before the failure, is called a *recovery* process [16].

An MH may become unavailable due to (i) mobile host disconnection, (ii) mobile host crash, and (iii) Access Bridge crash. As mentioned above, the MH suffers different permanent and transient failures and the storage on it is limited, so the storage is not suited to be treated as stable storage. Otherwise, an AB is on the border between wireless and wired network. In the Wireless CORBA architecture, all messages to and from an MH are traversed through ABs. Every message has a local copy in AB already, so it does not need to send an extra copy of each message elsewhere for logging purpose to tolerate the mobile host crash. So we choose the storage at AB as stable storage for the message logging and checkpointing protocol. The message logging mechanism in AB applies different methods for messages received from and sent to TB. AB logs messages after it receives them

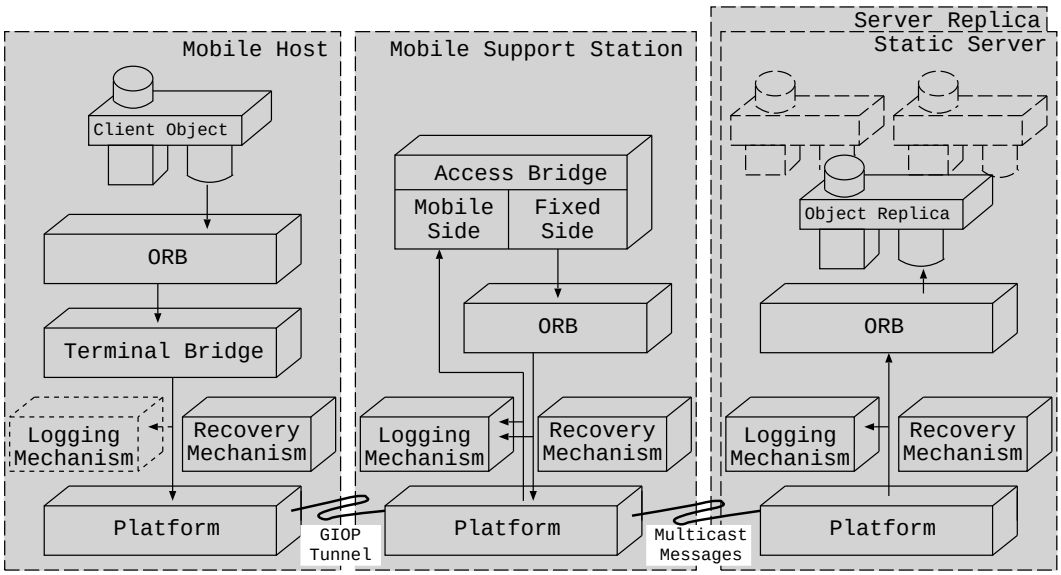



**Figure 2. Fault tolerant architecture overview**

from TB (receiver-based), but logs messages which is received from other hosts before it relays them to TB (quasi-sender-based). Quasi-sender-based message logging is not the sender but the intermediate proxy to log messages. TB may send back an acknowledgement depending on the received message's type.

The checkpointing strategy in an MH's is denoted as equi-number checkpointing. Equi-number checkpointing means checkpoints are equally placed with respect to the number of received messages $N$. It can be treated as two ways, one is the message number in each checkpointing interval is not changed and the other is the checkpoint number is not changed. We denote these two as equi-number checkpointing with respect to message number and equi-number checkpointing with respect to checkpoint number respectively. The first one is very naturally derived from the equidistant checkpointing strategy [16] as we change the required program execution time to the required message number. Because we choose the storage at AB as stable storage, after taking checkpoints, the data will be transmitted via wireless links and then saved at the current associated ABs. But the mobile computing environment does not restrict a user's location. When a user moves from one MSS to another, the carried MH should change its connected AB. So the location of the stable storage also would be changed during handoff [21]. It is one of the duties of our recovery protocol to find where the last checkpoint is located. An AB contains multiple associated TBs at the same time, but these TBs uses the AB only as a proxy, and there is no dependency between these TBs from the viewpoint of the AB . So an AB keeps different logs for different associated TBs.

A remote static server is replicated in *passive* or *active* style according the FT-CORBA standard. An AB communicates with remote static servers by a group communication system.It should detect and suppress duplicate

requests and replies, and deliver a single request or reply to the MH. The fault tolerance of HLA also can be achieved with the same schema. A lot of papers has discussed server replication and group communication [1, 6, 12, 13, 14, 22], so in this term paper we do not discuss static server's and HLA's fault tolerance in detail.

### 3.1. Data Structures

We employ the following data structures in our message logging and checkpointing mechanism.

- *Sequence Number (SN).* Each message exchanged in GIOP Tunnel has an SN, which identifies the message itself and the order in which the message is sent. An AB ensures the SN is distinct for a dedicated TB, but the SNs may be same between different TBs.

- *Message Record (MR).* There are two types of message record for two message logging mechanisms. The first type contains a message received by an AB from an TB and the status after the AB processes it. The second type includes an additional SN of the corresponding acknowledgement message, which indicates the order in which the message is received by an MH. The second type is used for messages sent to an TB.

- *CheckpointData* and *CheckpointDataReply* Messages. When an MH takes a checkpoint, it utilizes these two messages to reliably save the checkpoint in the current AB.

- *PurgeCheckpoint* Message. This message is sent out to clear old checkpoints when an AB receives a *CheckpointData* Message. It needs not be delivered reliably.

- *FetchCheckpoint* and *FetchCheckpointReply* Messages. A *FetchCheckpoint* Message is initialized when an MH detects a failure and starts to restore the state before the failure. A *FetchCheckpointReply* contains the last checkpoint of the MH.

### 3.2. Message Logging and Checkpointing

Let's see an event sequence in which a mobile client makes a request to a remote server when engaging pessimistic message logging. The steps, illustrated in Figure 3, are (1) A mobile client sends a request message *x* via a GIOP tunnel to the currently connected AB; (2) The AB logs *x* in the local stable storage pessimistically, sends an acknowledgement back to the client, and relays *x* to a remote server, then the AB waits for a reply; (3) After receiving the reply message *y*, the AB logs *y*, and dispatches it to the mobile client; (4) The mobile client sends a message back to acknowledge *y* and delivers *y* to the top level; (5) The AB logs the SN in the acknowledgement message with message *y*.

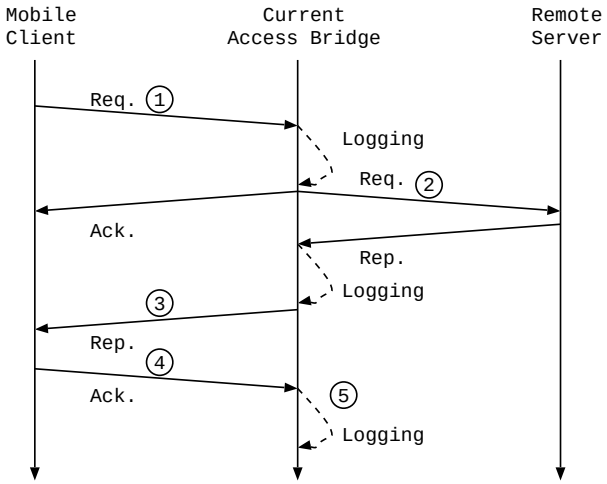



**Figure 3. Normal Operation Sequence**

When a TB receives a predefined number of messages since its last checkpoint, the TB will initialize a checkpointing procedure. The TB encapsulates the checkpoint in a *CheckpointData* message and sends this message to the current AB. To save wireless bandwidth, the checkpoint may not be sent out immediately, and it can be piggybacked with the next message from TB to AB [19]. The AB logs the checkpoint in its local stable storage and informs the HLA that a new checkpoint of the MH is saved in this AB. This information will be used in the recovery process to fetch the last checkpoint. The checkpointing interval is determined by the application requirements, the failure rate of the MH and the message arrival rate. It is also determined by the handoff frequency of the MH.

If an MH takes a checkpoint in the currently associated AB, the message logs and checkpoints prior to this checkpoint can be deleted since they are no longer necessary for recovery of the MH. So the AB will send a *PurgeCheckpoint* message to the HLA to delete those obsolete checkpoints and message records. This message needs not be reliably delivered, so long as any future *PurgeCheckpoint* message of the same MH will be delivered [9]. After the HLA receives the purge message, it will forward this request to the ABs in the itinerary track of the MH so that they can purge the unnecessary messages and collect the stable storage.

### 3.3. Mobile Host Handoff

In wireless networks which are organized in cells, a handoff is a mechanism for an MH to seamlessly change a connection from one AB to another. Handoff can be started due to two causes: normal operation and sudden connectivity loss. In normal operation, the MH will create a connection with a new AB. But in the second case, there is another successful outcome of the handoff procedure: connectivity re-established to the same AB as

before [17]. We identify two ABs in a handoff procedure with:

- Old Access Bridge (OAB) that was connected by an MH before the handoff.
- New Access Bridge (NAB) that would be connected after the handoff, which may be the same as the OAB.

Figure 4 depicts the handoff procedure where a mobile terminal re-establish connectivity to a new but different Access Bridge. In this figure, the MH creates a network connectivity (in network layer) with the NAB. Then it

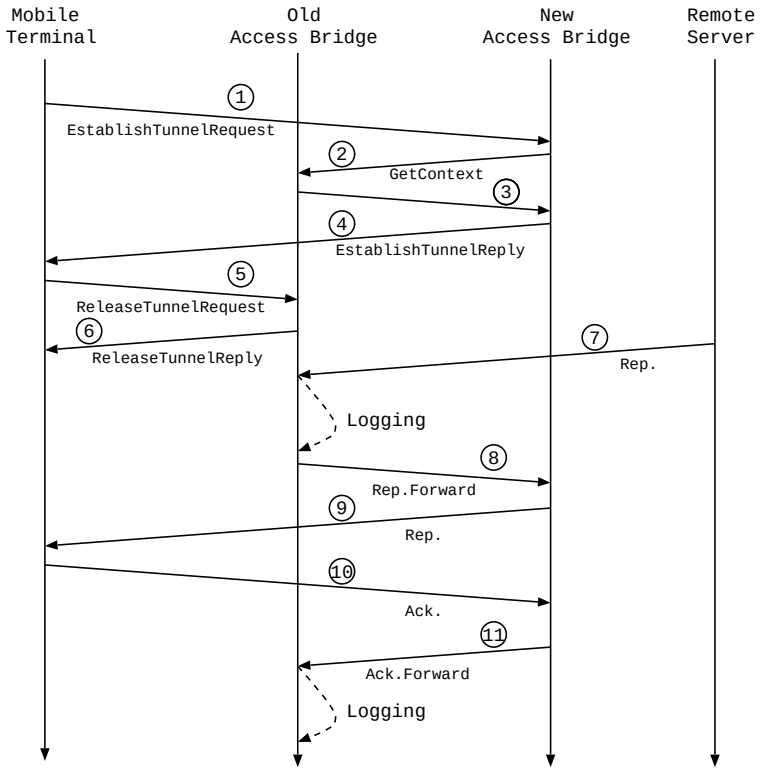



**Figure 4. Handoff Procedure**

sends a request message to establish a tunnel (message 1). The NAB uses information contained in the request message or acquired by querying the HLA to get the OAB of this MH. The NAB sends a message to the HLA to update this MH's location and invokes a handoff operation at the OAB (message 2). The OAB forwards necessary context data, such as *Sequence Number, Last Sequence Number Received, Connection ID*, to reconstruct the execution context in the NAB (message 3). The NAB sends the tunnel establishment reply to the MH (message 4) and the MH breaks the connection with the OAB (message 5 and 6). Afterwards, the MH sends and receives all messages through the NAB. The messages received by the OAB during the handoff (message 7), such as the replies to former requests, are forwarded to the NAB (message 8) and the NAB relays them to the MH (message 9). The acknowledgement messages (message 10) are forwarded to the OAB (message 11) to update the corresponding

message status in the logs to keep these message logs integrity. The GIOP requires that a reply should be sent in the same GIOP connection as the request came in [17]. So if a message is a reply for a request that was received through the OAB before the handoff, this message is encapsulated in a forward format and when the NAB receives it, the NAB should relay it to the OAB. All these forwarded messages are logged in the OAB.

### 3.4. Mobile Host Disconnection

An AB functions as a proxy between an MH and a static host. Its major function is to forward messages to and from the MH. We construct an AB with two parts (see Figure 5): Mobile side and Fixed side [17]. The mobile side connects with the MH by GIOP Tunnel, while the fixed side uses normal IIOP (Internet Inter-ORB Protocol) connections to communicate with remote static hosts. The AB keeps different maps between these two parts by *Connection ID* specified in [17] for every associated TB. Using the AB as a proxy, we can hide sudden mobile host disconnection from the remote host [26].

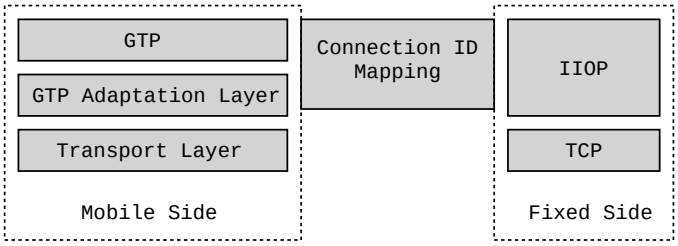



**Figure 5. Access Bridge ORB**

According to the fact that an AB is a proxy for relaying GIOP messages, we define three statuses of a message in an AB.

- *Received*. This is the default status for a message when an AB receives this message.
- *Sent*. When a message is relayed to an MH, a static host or another AB but before receiving the acknowledgement or the reply, the status of the message is *Sent*.
- *Processed*. After an AB receives an acknowledgement message or a reply, it changes the corresponding message's status to *Processed*.

If an AB receives a message which does not need to be relayed, the status of this message will be directly changed to *Processed* after the AB processes this message.

During a sudden mobile host disconnection, the last connected AB still keeps IIOP connections with remote hosts for a predefined time period. When the AB receives messages from the remote hosts, it logs messages but

does not forward them to the target MH (as message 7 in Figure 4) . When the AB gets a notification that the MH reconnects with the network, it forwards these received-but-not-sent messages to the MH (reconnects with the same AB) or the currently associated AB of this MH (reconnects with a new AB). If the MH recovers the connection with the same AB in a predefined time period, the AB will reuse these IIOP connections for succeeded communications. Otherwise the AB terminates all IIOP connections established for this MH. If the AB receives all the reply messages sent back from the remote host, it also closes these connections.

### 3.5. Mobile Host Crash

During disconnection, the state of the MH is kept intact. In case of mobile host crash, the local state is lost and needs to be recovered from the message logs. The MH is assumed to be fail-stop, i.e., the associated AB is able to detect the failure of the MH. Each failed MH can perform handoff and recovery procedure independently, which means that no other mobile hosts need to roll back together.

First the MH initiates a handoff procedure as depicted in Section 3.3. After the successful handoff, it starts a state recovery procedure. This procedure includes four phases:

1. The HLA finds the location of the last checkpoint and forwards it to the NAB;
2. The HLA collects all succeeded message records from the itinerary track of the MH and forwards them to the NAB;
3. The NAB sends the checkpoint, sorts the *processed* messages by their acknowledgement SNs, forwards these messages sequentially, and delivers the *sent* messages sequentially in their own SN order;
4. The MH initializes the application using the checkpoint and then executes the application. If a generated message has a counterpart message in the message record set, this message is inhibited and will not be sent out.

After applying the recovery procedure, the state of the MH will be restored to the state before failure. (We assume the application is deterministic.)

A GIOP tunnel is shared by all GIOP connections to and from the TB [17] , so some messages maybe arrive at the TB earlier than the messages sent before them. We adopt a quasi-sender-based message logging mechanism for these messages [9]. For reconstructing the same sequence of messages arriving before failure, we use the SNs of the corresponding acknowledgements in MRs to sort these processed messages before sending them out sequentially.

In our approach, if a user moves from one MSS to another, the stable storage for storing checkpoints and messages is changed accordingly. So if the mobile user traverses many times during a checkpointing period, the logged messages are scattered in these ABs. If we want to recover an MH from a failure or to revoke the stable storage for outdated messages, we need a method to find all these messages. Because the HLAs keep one itinerary track for each MH, we can employ these tracks to facilitate the messages collection and storage revocation, as described in Section 3.2.

The recovery period is time consuming because visiting different ABs is required to collect necessary message records. The reason is that when a failure occurs, the messages are scattered. We can improve this recovery procedure by collecting messages to a stable storage near or in the current AB. A strategy proposed in [21] ensures that the message logs and the checkpoint corresponding to the MH are at the "predecessor" AB. To achieve this, during handoff, a message is sent to the predecessor AB to transfer the checkpoint and logs. But if the MH moves frequently to another AB, this strategy will still create heavy volume of data transfer. We improve this strategy by letting the HLA trigger the transfer of the checkpoint and logs. In the HLA, there is a daemon and an array of timers for each MH. If one timer is expired, the daemon will dispatch a thread to handle the data transfer for the corresponding MH. The thread will collect the last checkpoint and successive message logs for this MH and save the data in the current AB of this MH. The timer is adaptive. It will extend the time period if an MH moves frequently and it will shorten the time period if the MH maintains connection with an AB for a long time. If a checkpoint is taken during this message collection period, the HLA stops the related thread. We also can use the number of handoff as the trigger of message collection.

### 3.6. Access Bridge Crash

An AB facilitates the connection mapping between an MH and a static host, so the AB is in the critical path between the MH and the static host. For tolerating an AB failure, normal replication strategy can be adopted. Recognizing the nomadic feature and the handoff mechanism in the mobile computing environment, we utilize a strategy that replicates the execution context and messages in an AB to its *Previous* AB (PAB) for each MH. An PAB for an MH is an AB in its movement track just one hop before its current AB. If there is no movement track for this MH, we choose the HLA as the "previous" AB. This strategy is passive. Some messages that do not change the status of the AB will not be replicated. Because each MH has different movement tracks, this strategy generates different AB replicas for different mobile hosts. After an AB failure, different mobile hosts can move to different NABs to start the handoff procedures.

If an AB crashes, all the associated MHs will detect this failure and then start handoff procedures independently.

If in the current location area there is only one MSS, the mobile user should explicitly move to another location for handoff. The handoff procedure has some differences from the normal handoff. The NAB first queries the HLA for the location of the replicated message logs and makes a request to the AB. The AB re-constructs the execution context from the message logs and sends this context to the NAB. The NAB initializes a new context for this MH according the received context and re-sends those messages which have no acknowledgments or replies to the MH and those target hosts. For saving the wireless bandwidth, the SNs in those messages to the MH are not in the vector which contains all the SNs of the messages received by this MH after the last checkpoint. After a successful handoff, the terminal informs the HLA that the recovery procedure is finished and continues to work as in normal condition. The HLA removes the failed AB from the MH track to avoid to select the failed AB as an PAB. If the MH moves back to the previous AB, the recovery procedure will be more efficient because all messages required to recovery are in the local storage. If the AB restarts after a failure, the MH can create connectivity with this AB just as a normal handoff from the PAB.

To avoid re-execute resent messages after an AB crash, a remote server should do some special work. When the server sends a reply message to the crashed AB, it learns that the AB is not reachable and then logs this reply message locally. After a successful handoff, the NAB reissues the same request through a new GIOP connection, the server identifies this request, retrieves the corresponding reply from its local log, and sends it back. Therefore the server does not process the same request more than once and keeps the data consistent.

## 4. Performance and Availability Analysis

In this section, we'll discuss the performance and availability of program executed on MH with and without engaging checkpointing. We do not take the effects of message logging into account.

### 4.1. Program Execution without Checkpointing

As mentioned above, GTP messages are transmitted through GIOP tunnels via wireless links during a program execution. These messages can be identified into two classes: control message and computational message. During different runs of an application, an MH will receive a given number of computational messages, but the number of control messages received may be varied as the failures and handoffs are random events. So let us consider the execution of a program in an MH with a given number of computational messages in the presence of failures and handoffs.

### 4.1.1 Model Description

Let $N$ denote the number of computational messages that a program in an MH should receive. The execution of this program in the presence of failures and handoffs is shown in Figure 6. After a failure, the program should be restarted from its beginning. All computation from its beginning to the failure is lost. These computation is denoted as wasted computation [20]. The program will eventually terminate successfully if it receives $N$ messages continuously before next failure. During a handoff, no computational message is sent to the MH. Only after a handoff, computational messages can be transmitted continuously.

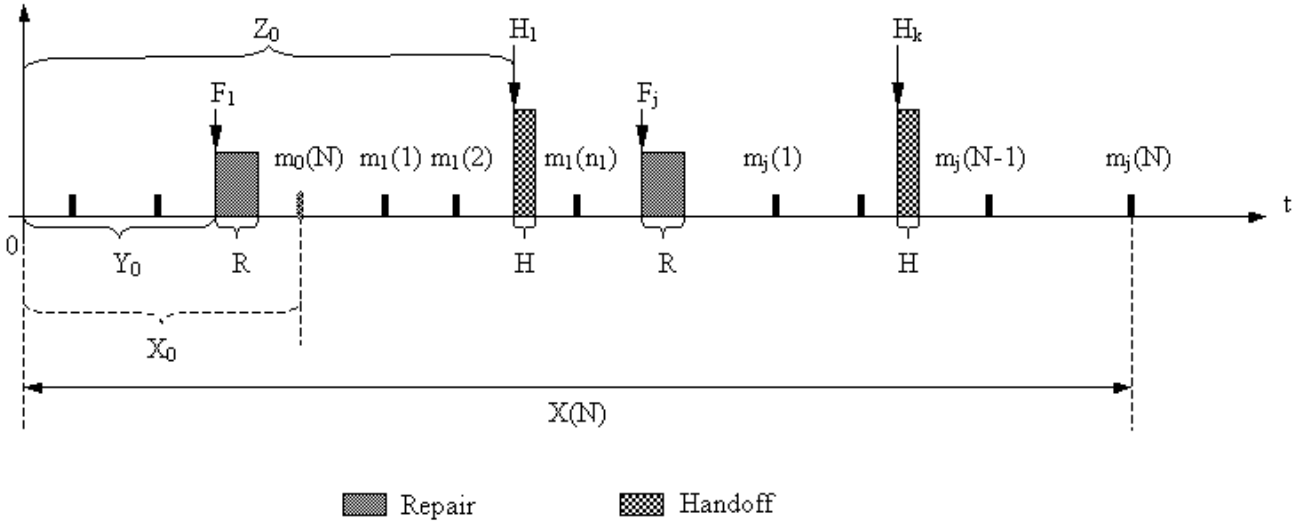



**Figure 6. Program execution without checkpointing**

We denote the $j^{th}$ failure, $j = 1, 2, \ldots$, and its time of occurrence by $F_j$ and $t(F_j)$ respectively, where $t(F_j) \geq 0$. Without loss of generality, we assume that the program starts processing at time $t = 0$ [24]. Let $Y_j$, $j = 1, 2, \ldots$, be the inter-failure time between $F_j$ and $F_{j+1}$, that is, $Y_j = t(F_{j+1}) - t(F_j)$ and $Y_0 = t(F_1)$. After a failure, there is a repair time $R$. The restarting time is not changed despite when the failure occurs, so we ignore it. We define $X_j$ as time period between the time when the repair is finished after $j^{th}$ failure and the time when the program receives $N$ messages if there is no failure afterwards. $X_0$ is the program execution time if there is no failure. We also denote the $k^{th}$ handoff, $k = 1, 2, \ldots$, and its time of occurrence by $H_k$ and $t(H_k)$, where $t(H_k) \geq 0$, respectively. Let $Z_k$, $k = 1, 2, \ldots$, be the inter-handoff time as $Z_k = t(H_{k+1}) - t(H_k)$ and $Z_0 = t(H_1)$. The handoff time is $H$. Additionally, the following assumptions are adopted [4]:

*Assumption 1.* The instants of the occurrences of the failures form a homogeneous Poisson process of parameter $\lambda$, i.e., $Y_j$, $j = 1, 2, \ldots$, are independent exponential random variables with parameter $\lambda$.

*Assumption 2.* Time between two successive message arrivals to a program is modeled as exponential distribu-

tion with parameter $\gamma$.

*Assumption 3.* Time between two successive handoff events is modeled as exponential distribution with parameter $\rho$.

*Assumption 4.* Failures do not occur when the program is in the repair process.

*Assumption 5.* A failure is detected as soon as it occurs.

### 4.1.2 Execution Time and Average Availability without Checkpointing

Under the above assumptions, the random variable $X_j$ has an N-stage Erlang distribution with parameter $\gamma$ [25]. Let $X(N)$ be the total program execution time with the $N$ computational messages.

The Laplace-Stieltjes Transform (LST) of the program execution time $X(N)$ has the form

$$\phi_X(s,N) = \frac{(s+\lambda)\gamma^N \left[\frac{1}{(s+\gamma+\lambda+\rho)^N} + \frac{1}{(s+\gamma+\lambda-\rho\ln\phi_H(s))^N} - \frac{1}{(s+\gamma+\lambda+\rho-\rho\ln\phi_H(s))^N}\right]}{(s+\lambda) - \lambda\phi_R(s)\left[1-\left(\frac{\gamma}{s+\gamma+\lambda}\right)^N\right]}, \tag{1}$$

and the expectation of program execution time $X(N)$ is

$$E(X(N)) = \left[\frac{1}{\lambda} + E(R)\right]\left[\left(\frac{\gamma+\lambda}{\gamma}\right)^N - 1\right] + \frac{\rho N \cdot E(H)}{\gamma+\lambda}\left[1-\left(\frac{\gamma+\lambda}{\gamma+\lambda+\rho}\right)^{N+1}\right]. \tag{2}$$

In the above equations, $\phi_R(s)$, $E(R)$, $\phi_H(s)$ and $E(H)$ denote the LST and the expectation of repair time and handoff time respectively.

A proof is given in Appendix A. Equation (2) shows that the expectation of program execution time is an exponential function of the number of messages $N$. It also depends on failure rate $\lambda$, message arrival rate $\gamma$ and handoff rate $\rho$. The second term at the right side of Equation (2) tends to 0 as $\rho$ tends to $0^+$, which implies the effect of handoffs could be ignored or the execution of a program will complete before the MH makes its first handoff. From this we know that the first term stands for the expected execution time if we do not take the effect of handoffs into account, in which $[1/\lambda + E(R)]$ denotes the expectation of inter-failure time and the following repair time, and $[(1+\lambda/\gamma)^N - 1]$ denotes the expected number of failure occurrence during executions. The second term stands for how handoffs influence the program execution time. We know that $[(\gamma+\lambda)/(\gamma+\lambda+\rho)]^{N+1}$ can be regarded as a constant when $N$ is large enough. So the second term is a linear function of the message number $N$, while the first term has an exponential relationship with $N$.

Average availability can be defined as how much of the time an MH is in uptime interval during an execution. An execution of a program in an MH may be decomposed into uptime and downtime intervals which occur in alternation [20]. If a program produces useful work towards its completion then it is in *uptime interval*, and a *downtime interval* is one in which the program does not produce useful work. The average availability is equal to

$uptime/(uptime + downtime)$. The times when the MH is in repair or in handoff are considered as downtime interval. If the MH is functional but the program is not producing useful computation, as mentioned as wasted computation, the time interval is also regarded as downtime interval. According to our definitions of uptime interval and downtime interval, the expected execution time of a program without failures and handoffs is the uptime and the expected execution time of a program in the presence of failures and handoffs is the sum of the uptime and downtime. So the average availability of a program without checkpointing which is required to receive $N$ messages, denoted as $A(N)$, is

$$A(N) = \frac{N}{\gamma \cdot E(X(N))}. \tag{3}$$

$A(N)$ decreases as message number $N$ or handoff arrival rate $\rho$ increases and as message arrival rate $\gamma$ decreases. As $N$ tends to $\lfloor 1/\lambda \rfloor$, $A(N)$ tends to 0, which means the program needs infinite long time to successful complete. So it needs some techniques to cut down the wasted computation time to increase the average availability. Checkpointing and rollback are such techniques which are suitable to achieve this object.

### 4.2. Equi-number Checkpointing

Equi-number checkpointing means checkpoints are equally placed with respect to the number of received messages $N$. It can be treated as two ways, one is the message number in each checkpointing interval is not changed and the other is the checkpoint number is not changed. We denote these two as equi-number checkpointing with respect to message number and equi-number checkpointing with respect to checkpoint number respectively. The first one is very naturally derived from the equidistant checkpointing strategy [16] as we change the required program execution time to the required message number.

#### 4.2.1 Model Description

The number of messages $N$ is divided into $n$ equal intervals, so each interval has $a = N/n$ messages. Despite of $N$, the checkpointing strategy will be equi-number checkpointing with respect to checkpoint number if $n$ is fixed and be equi-number checkpointing with respect to message number if $a$ is fixed. Actually these two approaches have the same mathematical expression of the expected execution time and average availability. So afterwards we only consider equi-number checkpointing with respect to message number. For simplicity, we take a checkpoint even when a program finishes. Therefore there is always a checkpoint at the end of each interval, thus a total of $n$ checkpoints. The execution times of the $n$ intervals are independent and identically distributed random variables. In each interval, the execution is the same as the execution without failures except that the program restarts from its most recent checkpoint. Figure 7 shows the execution with equi-number checkpointing in the $i^{th}(0 \leq i < n)$

interval, in which $X_i(j), Y_i(j), Z_i(j), j = 0, 1, 2, \ldots$, denote the same events as $X(j), Y(j), Z(j), j = 0, 1, 2, \ldots$, in Figure 6.

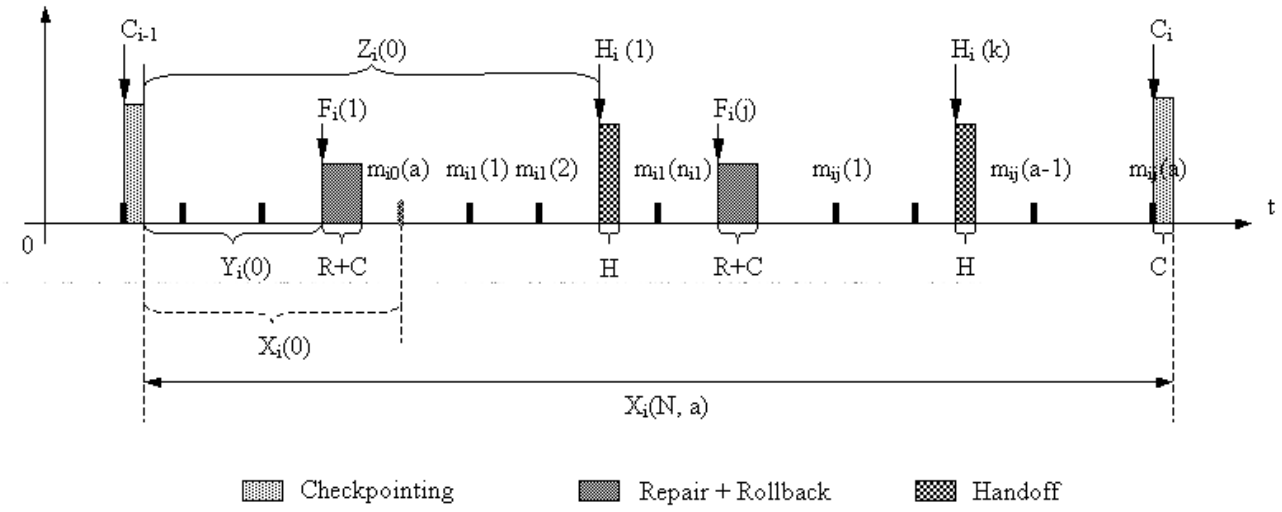



**Figure 7. Program execution with equi-number checkpointing**

Let $C$ denote the checkpoint creation time. We know $C$ contains two parts of time, which are the time to take a checkpoint and the time to save the checkpoint on stable storage. In our mobile model, as we know, the MH is not safe to be treated as stable storage. Checkpoints should be transmitted through a wireless link, which suffers with little bandwidth and long transfer delay, and then be saved on the stable storage of currently associated AB. So the latter part is the most significant one. After a failure, a recent checkpoint should be reloaded into the MH through a wireless link. As the same reason, the main part of the rollback time is consumed by checkpoint transmission. So we let the rollback time also be $C$. We adopt one more assumption that failures do not occur when the program is in the rollback process.

### 4.2.2 Execution Time and Average Availability with Checkpointing

Let $X(N, a)$ denote the total execution time of a program which is divided into $N/a$ intervals, each interval should receive $a$ messages continuously without failure. The LST of $X(N, a)$ is given by

$$\phi_X(s, N, a) = \left[ \frac{(s+\lambda)\gamma^a \left[ \frac{\phi_C(s+\lambda+\rho)}{(s+\gamma+\lambda+\rho)^a} + \frac{\phi_C(s+\lambda)}{(s+\gamma+\lambda-\rho\ln\phi_H(s))^a} - \frac{\phi_C(s+\lambda+\rho)}{(s+\gamma+\lambda+\rho-\rho\ln\phi_H(s))^a} \right]}{(s+\lambda) - \lambda\phi_R(s)\phi_C(s)\left[1 - \left(\frac{\gamma}{s+\gamma+\lambda}\right)^a \phi_C(s+\lambda)\right]} \right]^{N/a}, \quad (4)$$

and the expectation of program execution time is

$$E(X(N,a)) = \frac{N}{a}\left[\frac{1}{\lambda} + E(R) + E(C)\right]\left[\phi_C(-\lambda)\left(\frac{\gamma+\lambda}{\gamma}\right)^a - 1\right]$$

$$+\frac{\rho N\cdot E(H)}{\gamma+\lambda}\left[1-\phi_C(\rho)\left(\frac{\gamma+\lambda}{\gamma+\lambda+\rho}\right)^{a+1}\right]. \tag{5}$$

A proof is given in Appendix B. Equation (5) shows that $E(X(N,a))$ is a linear function of the number of messages $N$ and an exponential function of the number of messages $a$ in each interval. In Equation (5), the first term also stands for the expected execution time if we do not consider the effect of handoffs, in which $N/a$ denotes the number of checkpointing intervals during an execution. The other terms have similar expressions with Equation (2). With checkpointing, the expectation of program execution time decreases dramatically.

In the program execution utilizing checkpointing, if we treat the checkpoint creation time and the time to rollback to the most recent checkpoint also as downtime, then the average availability $A(N,a)$ can be computed from

$$A(N,a)=\frac{N}{\gamma\cdot E(X(N,a))}, \tag{6}$$

which has the same form with Equation (3). With equi-number checkpointing which respects to message number, $A(N,a)$ is a constant and does not vary with $N$. But it still changes with $N$ in equi-number checkpointing with respect to checkpointing number, which has the same curve shape with $A(N)$.

We know that the benefit of checkpointing comes at a price. During failure free execution, checkpointing delays the message delivery. After a failure, there is a time to reload the program status in a checkpoint. So there exists an optimal checkpointing interval which minimizes the total program execution time or maximizes the average availability. The optimal checkpointing interval $\hat{a}$ can be obtained by solving $(\partial/\partial a)[E(X(N,a))]=0$ or $(\partial/\partial a)[A(N,a)]=0$ . The second term in Equation (5) can be regarded as a constant when $a$ is large enough and is insignificant compared with the first term , so we ignore it. The derived equation is

$$\left(\frac{\gamma+\lambda}{\gamma}\right)^{a}\left[1-a\ln\left(\frac{\gamma+\lambda}{\gamma}\right)\right]=\phi_C(\lambda). \tag{7}$$

Using the expansion of $[(\gamma+\lambda)/\gamma]^a$ as far as the second degree term, the approximate solution of the equation above is

$$\hat{a}\cong\left\lfloor\frac{\sqrt{2[1-\phi_C(\lambda)]}}{\ln\left(\frac{\gamma+\lambda}{\gamma}\right)}\right\rfloor, \tag{8}$$

from which we know that $\hat{a}$ is independent of the required message number $N$, but it depends not only on the failure rate $\lambda$ and the message arrival rate $\gamma$, but also on the distribution of checkpoint duration. The optimal checkpoint count is $\hat{n}=\lfloor N/\hat{a}\rfloor$. We denote the minimal value of the expected program execution time and the maximal value of the average availability as $E^*(X(N,a))$ and $A^*(N,a)$ respectively when $a=\hat{a}$. Compared with the execution without checkpointing, the execution with equi-number checkpointing is beneficial only if

$E(X(N,a)) < E(X(N))$, the approximate solution is given by

$$N > 2 \left\lfloor \frac{\phi_C(-\lambda)\left(\frac{\gamma+\lambda}{\gamma}\right)^a - 1}{a\left[\ln\left(\frac{\gamma+\lambda}{\gamma}\right)\right]^2} - \frac{1}{\ln\left(\frac{\gamma+\lambda}{\gamma}\right)} \right\rfloor. \quad (9)$$

### 4.3. Simulations and Comparisons

To verify the correctness of the expected execution time with and without checkpointing we derived, we made two simulations respectively. In these two simulations, we considered the following values:

- mean message arrival rate $\gamma = 10^{-1}$;
- mean failure arrival rate $\lambda = 10^{-3}, 10^{-4}, 10^{-5}$;
- mean checkpoint creation time $E(C) = 1$;
- checkpoint count $N/a$ is fixed and is 10;
- mean repair time $E(R) = 10$;
- mean handoff rate $\rho = 10^{-3}$ and mean handoff time $E(H) = 10$.

We run each simulation 50 times given different failure arrival rates and the results of the mean execution time are shown in Figure 8. These two figures show that the simulations match the curves of derived expected execution time. To see how the failure rate influences the program execution time, we depict Y axis as the ratio of the expectation of execution time with failure to the expectation without failure. In each case, the curves increase exponentially and have the same shape as the number of messages required increases. Figure 8(a) and 8(b) are similar, except that the labels of X axis. The difference between them is exact the value of $N/a$. We know that in each checkpointing interval, the execution process is similar with the process without checkpointing. From another point, checkpointing decreases the failure arrival rate to $1/(N/a)$ of the failure arrival rate without checkpointing .

Figure 9 shows the result of average availability as we change the value of checkpoint count $N/a$. From this graph, we see that for the given message arrival rate $\gamma$, failure arrival rate $\lambda$ and handoff rate $\rho$, there is a descend of average availability when the required message count decreases to a certain value. The larger the $N/a$ is, the greater the transition point is. This is to say that under some conditions, more checkpoints bring more overhead than we expect. Certainly, as the required message number increases, more checkpoints will be more beneficial. The dashed curve represents the average availability without checkpointing as $\lambda = 10^{-5}$, which is always above

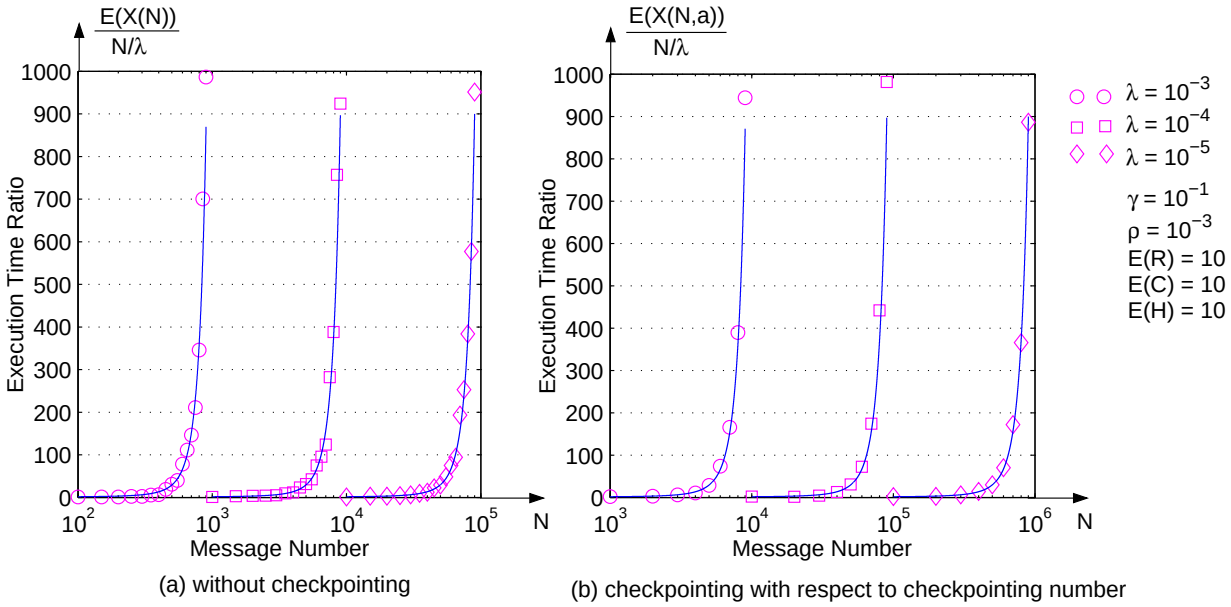



**Figure 8. Simulation result without checkpointing and with checkpointing**

the curve of checkpointing as $\lambda = 10^{-4}$ and $N/a = 10$. Because there are some overheads in checkpointing, such as checkpoint creation time, etc. We treat these overheads as downtime. Another difference is there is no descend of average availability without checkpointing as the required message number decreases. It hints that checkpointing is not always beneficial to the program execution time or the average availability. We will see it in Figure 10.

To illustrate whether the checkpointing can be used to reduce the overall execution time or increase the average availability, we compare the average availability time with and without checkpointing from another viewpoint. This time we utilize equi-number checkpointing with respect to message number, i.e., the number of messages required $a$ is a constant in each checkpointing interval, under which the average availability is a constant, shown as horizontal lines in Figure 10. The average availability increases as $a$ decreases, which means more checkpoints are to be taken in execution. While excessive checkpointing decreases the availability, as shown in Figure 11. These horizontal lines have intersections with those curves which represent no checkpoint is taken during execution. Naturally, under these intersections, a program can be run more economically by not taking checkpoints and by rerunning from the beginning of the program [7].

Figure 11 shows that when we engage equi-number checkpointing with respect to message number, there exists an optimized $\hat{a}$ maximizing the average availability. The approximation solution of $\hat{a}$ is expressed in Equation (8). Under the parameters provided in this figure, $\hat{a} = 44$ and $A^*(N, a) = 0.951$. With equi-number checkpointing

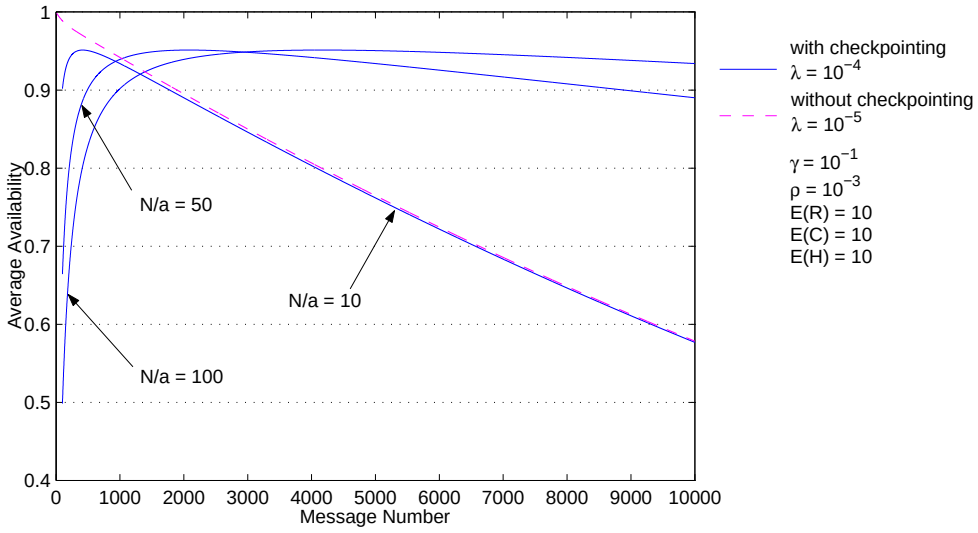



**Figure 9. Equi-number checkpointing with respect to checkpoint number**

which respects to checkpoint number $n$ and $n = 10$, the availability gets maximal value as $N = 440$. Under $N = 92$, checkpointing can not get any benefit on the availability.

Figure 12 demonstrates the variation of average availability with message arrival rate $\gamma$ and handoff rate $\rho$ when engaging checkpointing with respect to checkpoint number or not. The average availability decreases as $\rho$ increases, despite engaging checkpointing or not. The availability increases as $\gamma$ increases when the program executes without checkpointing. But with checkpointing which respects to checkpoint number, the availability increases first and then it decreases. When message arrival rate is low, checkpointing increases the availability. But when message arrival rate is high, the program will be completed by experiencing less failures. Most of the checkpoints do not contribute to the availability. So checkpointing incurs overhead more seriously. Under this condition, we should take checkpoints less frequently to reduce these overheads.

Figure 9– 12 demonstrate that there are several factors to be weighted before determining that any checkpoint is worth taking [7], such as failure arrival rate, message arrival rate, handoff rate, checkpoint creation cost, message number, etc. So if we can adaptively adjust the checkpointing parameters and choose among checkpointing with respect to checkpoint number, checkpointing with respect to message number and no checkpointing, we will get more performance improvement.

## 5. Summary and Future Work

This term paper discusses how to engaging checkpointing and message logging in Wireless CORBA firstly. It employs both quasi-sender-based and receiver-based message logging methods. The protocol can tolerate mobile

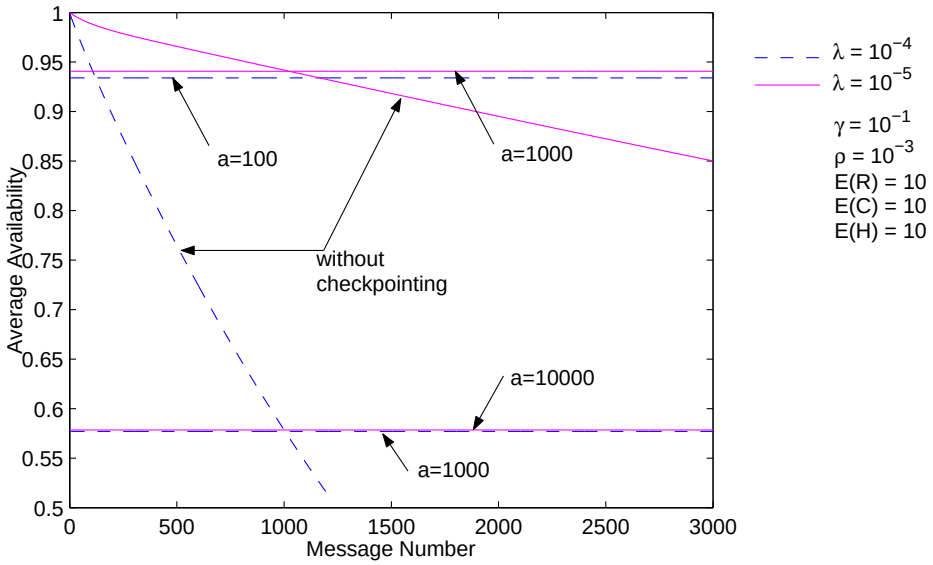



**Figure 10. Equi-number checkpointing with respect to message number**

host disconnection, mobile host crash and Access Bridge crash. It introduces two equi-number checkpointing strategies which respects to checkpoint number and message number respectively. It chooses the storage available at the Access Bridge as the stable storage to log messages and checkpoints. To tolerate the Access Bridge crash, it replicates an Access Bridge's state in the previous Access Bridge for each mobile host. It also engages the handoff mechanism as a means to recover from the Access Bridge crash. Then it analyzes the program execution time and average availability with and without checkpointing, which is under the assumption that the number of total received computational message $N$ is not changed during a run and the inter-failure arrival time, the inter-message arrival time and the inter-handoff time are exponentially distributed. The program execution time is an exponential function of $N$ without checkpointing and is a linear function of $N$ with checkpointing. It has a linear relationship with handoff rate despite engaging checkpointing or not. We also showed that under certain conditions, checkpointing incurs worse performance than no checkpointing. So the individual programs are best off by adopting checkpointing suited to these parameters or not.

In fault tolerant wireless CORBA, we assume faults have been detected already. Actually we should think about how to detect a fault. A fault detector is an oracle that provides MHs with information on the behavior of the other MHs or SHs. Within a distributed system dominated by uncertainty, such as asynchrony, failures etc., we need a consensus to provide correct host with a single view of the system. The hosts may attempt to agree on whether to classify a host as faulty. Mobile environments are often subject to link failures, mobile host failures, and MHs may be disconnect intermittently. So consensus problems are more difficult to solve. Additionally, the algorithm

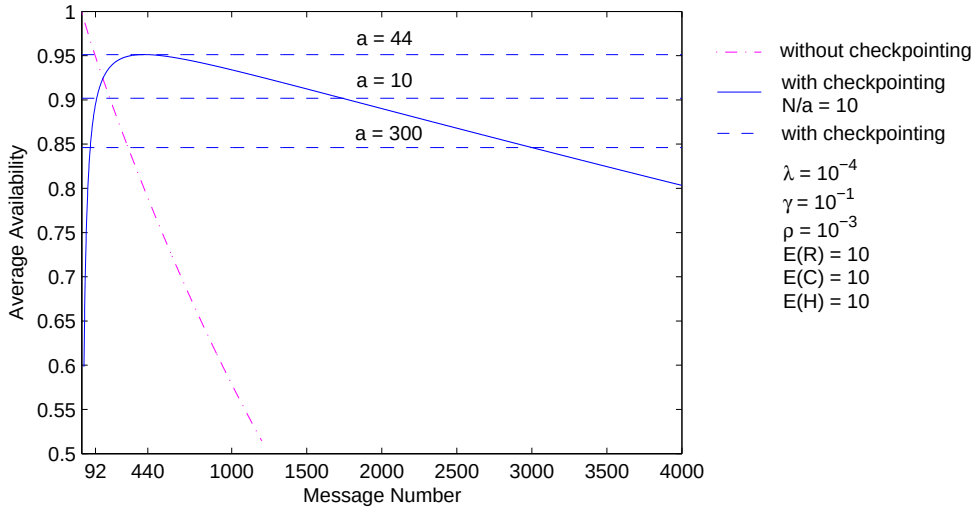
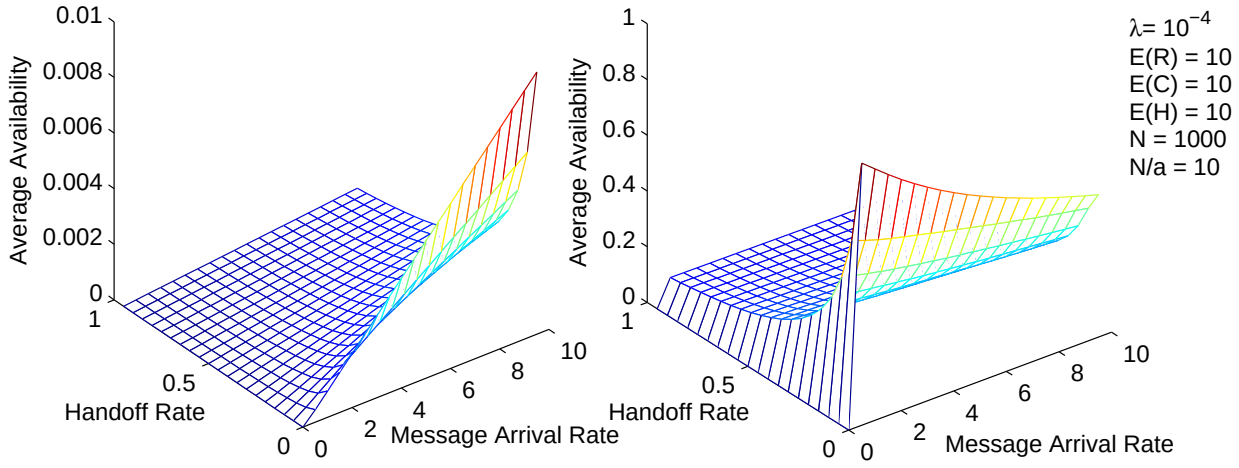



**Figure 11. Comparison between checkpointing and without checkpointing**

should lead to a faster solution and reduce communication cost. In our performance analysis model, we assume that after a failure the messages still arrive as normal situation with exponential distribution. But during repair, rollback and recovery interval, the messages are queued in ABs. So after a failure, there should be a time interval in which the time between two successive message arrivals is not distributed exponentially. We should take this into account to make the analysis model more realistic.

How to provide fault tolerance in ad hoc network is another future work. Ad hoc network is a collection of MHs without the need of infrastructure support. There is not stable storage can be used as the wireless CORBA. Ad hoc network is self-organizing and adaptive. All these are challenges to fault tolerance.




(a) without checkpointing

(b) checkpointing with repsect to checkpointing number

**Figure 12. Average availability vs. message arrival rate and handoff rate**

## Appendix[1]

## A Proof of the program execution time without checkpointing

If $X_0 = x$, $Y_0 = y$ and $Z_0 = z$, we have

$$X(N)|_{X_0=x,Y_0=y,Z_0=z} = \begin{cases} x & : \ if \ \ x \leq y \ and \ x \leq z \\ x + \rho x H & : \ if \ \ x \leq y \ and \ x > z \\ y + R + X(N) & : \ if \ \ x > y. \end{cases} \tag{10}$$

If $x \leq y \ and \ x \leq z$, then the program will complete in $x$ units of time without failures and handoffs. If $x \leq y \ and \ x > z$, then the program will make $\rho x$ handoffs on average before it receives $N$ messages without failures. In this case, there is an expected handoff time $\rho x H$. If $x > y$, then a failure occurs before the program receives $N$ messages. In this case, there is a repair time $R$ after which the program execution is restarted from its beginning, which means that the program is required to receive $N$ messages without interrupt again [16].

The LST of (10) can be written as

$$\phi_X(s,N)|_{X_0=x,Y_0=y,Z_0=z} = \begin{cases} e^{-sx} & : \ if \ \ x \leq y \ and \ x \leq z \\ e^{-sx}[\phi_H(s)]^{\rho x} & : \ if \ \ x \leq y \ and \ x > z \\ e^{-sy}\phi_R(s)\phi_X(s,N) & : \ if \ \ x > y. \end{cases} \tag{11}$$

[1]The proofs in Appendix A and B follow the same method in [16].

Unconditioning on $X_0$, $Y_0$ and $Z_0$, we get

$$\begin{aligned}\phi_X(s,N) &= \int_{x=0}^{\infty}\int_{y=0}^{\infty}\int_{z=0}^{\infty}\phi_X(s,N)|_{X_0=x,Y_0=y,Z_0=z}\cdot\frac{\gamma^N x^{N-1}e^{-\gamma x}}{(N-1)!}\cdot\lambda e^{-\lambda y}\cdot\rho e^{-\rho z}dzdydx \\ &= \gamma^N\left[\frac{1}{(s+\gamma+\lambda+\rho)^N}+\frac{1}{(s+\gamma+\lambda-\rho\ln\phi_H(s))^N}-\frac{1}{(s+\gamma+\lambda+\rho-\rho\ln\phi_H(s))^N}\right] \\ &\quad+\frac{\lambda\phi_R(s)\phi_X(s,N)}{s+\lambda}\left[1-\frac{\gamma^N}{(s+\gamma+\lambda)^N}\right] \qquad (12)\end{aligned}$$

From (12), we have

$$\phi_X(s,N)=\frac{(s+\lambda)\gamma^N\left[\frac{1}{(s+\gamma+\lambda+\rho)^N}+\frac{1}{(s+\gamma+\lambda-\rho\ln\phi_H(s))^N}-\frac{1}{(s+\gamma+\lambda+\rho-\rho\ln\phi_H(s))^N}\right]}{(s+\lambda)-\lambda\phi_R(s)\left[1-\left(\frac{\gamma}{s+\gamma+\lambda}\right)^N\right]} \qquad (13)$$

Using the relations $E(X(N))=-\frac{\partial\phi_X(s,N)}{\partial s}|_{s=0}$, $E(R)=-\frac{d\phi_R(s)}{ds}|_{s=0}$, $\phi_R(0)=1$, $E(H)=-\frac{d\phi_H(s)}{ds}|_{s=0}$, and $\phi_H(0)=1$ [3], the expected program execution time in the presence of failures and handoffs is given by

$$E(X(N))=\left[\frac{1}{\lambda}+E(R)\right]\left[\left(\frac{\gamma+\lambda}{\gamma}\right)^N-1\right]+\frac{\rho N\cdot E(H)}{\gamma+\lambda}\left[1-\left(\frac{\gamma+\lambda}{\gamma+\lambda+\rho}\right)^{N+1}\right] \qquad (14)$$

## B Proof of the program execution time with equi-number checkpointing

First we only consider the program execution time in $i^{th}$ interval. In each interval, we can use the same procedure in Appendix A. If $X_i(0)=x$, $Y_i(0)=y$ and $Z_i(0)=z$, we have

$$X_i(N,a)|_{X_i(0)=x,Y_i(0)=y,Z_i(0)=z}=\begin{cases}x+C & : \ if\ \ x+C\le y\ and\ x+C\le z\\ x+C+\rho xH & : \ if\ \ x+C\le y\ and\ x+C>z\\ y+R+C+X_i(a) & : \ if\ \ x+C>y.\end{cases} \qquad (15)$$

If $x+C\le y\ and\ x+C\le z$, then the program will complete the $i^{th}$ interval in $x+C$ units of time without failures and handoffs. If $x\le y\ and\ x>z$, then the program will make $\rho x$ handoffs on average before it receives $a$ messages without failures. In this case, there is an expected handoff time $\rho xH$. If $x+C>y$, then a failure occurs before the program receives $a$ messages. In this case, there is a repair time $R$ and a state restore time $C$ after which the program execution is restarted from its current interval's beginning.

The LST of (15) can be written as

$$\phi_{X_i}(s,N,a)|_{X_i(0)=x,Y_i(0)=y,Z_i(0)=z}=\begin{cases}e^{-sx}\phi_C(s) & : \ if\ \ x+C\le y\ and\ x+C\le z\\ e^{-sx}\phi_C(s)[\phi_H(s)]^{\rho x} & : \ if\ \ x+C\le y\ and\ x+C>z\\ e^{-sy}\phi_R(s)\phi_C(s)\phi_{X_i}(s,a) & : \ if\ \ x+C>y.\end{cases} \qquad (16)$$

Unconditioning on $X_i(0), Y_i(0)$ and $Z_i(0)$, we get

$$\begin{aligned}\phi_{X_i}(s,N,a) &= \int_{x=0}^{\infty}\int_{y=0}^{\infty}\int_{z=0}^{\infty}\phi_{X_i}(s,N,a)|_{X_i(0)=x,Y_i(0)=y,Z_i(0)=z}\cdot\frac{\gamma^a x^{a-1}e^{-\gamma x}}{(a-1)!}\cdot\lambda e^{-\lambda y}\cdot\rho e^{-\rho z}dzdydx \\ &= \gamma^a\left[\frac{\phi_C(s+\lambda+\rho)}{(s+\gamma+\lambda+\rho)^a}+\frac{\phi_C(s+\lambda)}{(s+\gamma+\lambda-\rho\ln\phi_H(s))^a}-\frac{\phi_C(s+\lambda+\rho)}{(s+\gamma+\lambda+\rho-\rho\ln\phi_H(s))^a}\right] \\ &\quad+\frac{\lambda\phi_R(s)\phi_C(s)\phi_{X_i}(s,N,a)}{s+\lambda}\left[1-\frac{\gamma^a\phi_C(s+\lambda)}{(s+\gamma+\lambda)^a}\right] \qquad (17)\end{aligned}$$

From (17), we have

$$\phi_{X_i}(s,N,a)=\frac{(s+\lambda)\gamma^a\left[\frac{\phi_C(s+\lambda+\rho)}{(s+\gamma+\lambda+\rho)^a}+\frac{\phi_C(s+\lambda)}{(s+\gamma+\lambda-\rho\ln\phi_H(s))^a}-\frac{\phi_C(s+\lambda+\rho)}{(s+\gamma+\lambda+\rho-\rho\ln\phi_H(s))^a}\right]}{(s+\lambda)-\lambda\phi_R(s)\phi_C(s)\left[1-\left(\frac{\gamma}{s+\gamma+\lambda}\right)^a\phi_C(s+\lambda)\right]} \qquad (18)$$

The total execution time $X(N,a)$ is the sum of $N/a$ independent random variables $X_i$, so the Laplace form is

$$\phi_X(s,N,a)=[\phi_{X_i}(s,N,a)]^{N/a} \qquad (19)$$

Using the relations $E(X_i(N,a))=-\frac{\partial\phi_{X_i}(s,N,a)}{\partial s}|_{s=0}$, $E(R)=-\frac{d\phi_R(s)}{ds}|_{s=0}$, $\phi_R(0)=1$, $E(H)=-\frac{d\phi_H(s)}{ds}|_{s=0}$, $\phi_H(0)=1$, $E(C)=-\frac{d\phi_C(s)}{ds}|_{s=0}$, and $\phi_C(0)=1$, the expected program execution time of the $i^{th}$ interval in the presence of failures and handoffs is given by

$$E(X_i(N,a))=\left[\frac{1}{\lambda}+E(R)+E(C)\right]\left[\phi_C(-\lambda)\left(\frac{\gamma+\lambda}{\gamma}\right)^a-1\right]+\frac{\rho a\cdot E(H)}{\gamma+\lambda}\left[1-\phi_C(\rho)\left(\frac{\gamma+\lambda}{\gamma+\lambda+\rho}\right)^{a+1}\right] \qquad (20)$$

Finally, we have that

$$\begin{aligned}E(X(N,a)) &= \frac{N}{a}E(X_i(N,a)) \\ &= \frac{N}{a}\left[\frac{1}{\lambda}+E(R)+E(C)\right]\left[\phi_C(-\lambda)\left(\frac{\gamma+\lambda}{\gamma}\right)^a-1\right] \\ &\quad+\frac{\rho N\cdot E(H)}{\gamma+\lambda}\left[1-\phi_C(\rho)\left(\frac{\gamma+\lambda}{\gamma+\lambda+\rho}\right)^{a+1}\right] \qquad (21)\end{aligned}$$

## References


[1] M. Barborak, M. Malek, and A. Dahbura. The consensus problem in fault-tolerant computing. *ACM Computing Sruverys*, 25(2):171–220, June 1993.

[2] X. Chen and M. R. Lyu. Message logging and recovery in Wireless CORBA using access bridge. *The 6th International Symposium on Autonomous Decentralized Systems*, April 2003.

[3] D. R. Cox. *Renewal Theory*. Methuen & Co Ltd., London, 1962.

[4] A. Duda. The effects of checkpointing on program execution time. *Information Processing Letters*, 16:221–229, June 1983.

[5] M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408, September 2002.

[6] P. Felber, B. Garbinato, and R. Guerraoui. The design of a CORBA group communication service. *The 15th Symposium on Reliable Distributed Systems*, pages 150 –159, October 1996.

[7] D. P. Jasper. A discussion of checkpoint/restart. *Software Age*, 3(10):9–14, October 1969.

[8] J. Jing, A. Helal, and A. Elmagarmid. Client-server computing in mobile environments. *ACM Computing Surveys*, 31(2):117–156, June 1999.

[9] D. B. Johoson. Distributed system fault tolerance using message logging and checkpointing. *Ph.D. Dissertation, Rice University*, December 1989.

[10] P. Krishna, N. H. Vaidya, and D. K. Pradhan. Recovery in distributed mobile environments. *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 83–88, October 1993.

[11] Y. Ling, J. Mi, and X. Lin. A variational calculus approach to optimal checkpoint placement. *IEEE Transactions on Computers*, 59(7):699–708, July 2001.

[12] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, April 1996.

[13] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. A fault tolearnce framework for CORBA. *The 29th International Symposium on Fault-Tolerant Computing*, pages 150–157, June 1999.

[14] P. Narasimhan, K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. Providing support for survivable CORBA applications with the immune system. *The 19th IEEE International Conference on Distributed Computing Systems*, pages 507–516, May 1999.

[15] N. Neves and W. K. Fuchs. Adaptive recovery for mobile environments. *Communications of the ACM*, 40(1):68–74, January 1997.

[16] V. F. Nocola. Checkpointing and the modeling of program execution time. *Software Fault Tolerance, edited by Michael R. Lyu,John Wiley & Sons Ltd.*, pages 167–188, 1995.

[17] Object Management Group. Telecom wireless CORBA. *OMG Doucment dtc/01-06-02*, June 2001.

[18] Object Management Group. The Common Object Request Broker: Architecture and specification, 2.6.1 edition. *OMG Document formal/02-05-15*, May 2002.

[19] T. Park and H. Y. Yeom. An asynchronous recovery scheme based on optimistic message logging for the mobile computing systems. *The 20th International Conference on Distributed Computing Systems*, pages 436–443, April 2000.

[20] J. S. Plank and M. G. Thomason. The average availability of uniprocessor checkpointing systems, revisited. *Technical Report UT-CS-98-400, Dept. of Computer Science, Univ. of Tennessee*, August 1998.

[21] D. K. Pradhan, P. Krishna, and N. H. Vaidya. Recovery in mobile environments: Design and trade-off analysis. *The 26th International Symposium on Fault-Tolerant Computing*, June 1996.

[22] R. V. Renesse, K. P. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[23] R. Ruggaber and J. Seitz. Using CORBA applications in nomadic environment. *The 3rd IEEE Workshop on Mobile Computing Systems and Applications*, pages 161–170, December 2000.

[24] A. N. Tantawi and M. Ruschitzka. Performance analysis of checkpointing strategies. *ACM Transactions on Computer Systems*, 2(2):123–144, June 1984.

[25] K. S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Science Applications, 2nd edition*. John Wiley & Sons Ltd., New York, 2002.

[26] B. Yao and W. K. Fuchs. Proxy-based recovery for applications on wireless hand-held devices. *The 19th IEEE Symposium on Reliable Distributed Systems*, pages 2–10, October 2000.