Performance and Effectiveness Analysis of Checkpointing in Mobile Environments

Xinyu Chen and Michael R. Lyu
Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
{*xychen, lyu*}*@cse.cuhk.edu.hk*

Abstract

Many mathematical models have been proposed to evaluate the execution performance of an application with and without checkpointing in the presence of failures. They assume that the total program execution time without failure is known in advance, under which condition the optimal checkpointing interval can be determined. In mobile environments, application components are distributed and tasks are computed by sending and receiving computational and control messages. The total execution time includes communication time and depends on multiple factors, such as heterogeneous processing speeds, link bandwidth, etc., making it unpredictable during different executions. However, the number of total computational messages received is usually unchanged within an application. Another special factor that should be considered for checkpointing purpose is handoff, which often happens in mobile networks. With these observations, we analyze application execution performance and average effectiveness, and introduce an equi-number checkpointing strategy. We show how checkpointing and handoff affect performance and effectiveness metrics, determine the conditions when checkpointing is beneficial, and calculate the optimal checkpointing interval for minimizing the total execution time and maximizing the average effectiveness in mobile environments.

Keywords: *Performance analysis, Equi-number checkpointing, Optimal checkpointing interval, Mobile environments, Handoff*

1. Introduction

According to advances of wireless networking technologies, many portable information appliances are widely available, bringing about a computing paradigm shift in the direction of nomadic computing [2]. Nomadic computing enables users to access and exchange information while they roam around in mobile environments. This flexibility, however, causes more probable physical damage to mobile hosts (MHs) [15]. In addition, MHs have low battery power and wireless links suffer limited bandwidth and long transfer delay, making transient failures more likely. Thus, nomadic computing requires techniques to provide fault tolerance for continuing services despite of such failures.

Checkpointing and rollback is among the best-known techniques to minimize loss of computation in the presence of failures by periodically saving the programs' states on stable storage during failure-free execution. Each of the saved states is called a *checkpoint* [8], 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 until the point just before the failure, is called a *recovery* process [16].

If we engage the checkpointing 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 with a price. Excessive checkpointing would result in performance degradation, while deficient checkpointing would still incur an expensive recovery overhead [14]. Therefore, a trade-off exists. Many mathematical models have been proposed to evaluate the execution performance and to derive the optimal checkpointing interval [7, 16, 20, 23]. But these models assume that the total program execution time with no failure is known in advance. In mobile environments, application components are distributed and tasks are completed by sending and receiving computational and control messages. The total execution time is dependent on multiple factors, such as heterogeneous components and processing powers. Moreover, as part of the total application execution time, messages passing time is affected by link bandwidth, which varies during different executions. However, the number of total computational messages received is usually not changed for an application. Furthermore, we need to consider handoff situation, which often happens in mobile networks. In this paper, we use these observations to analyze the application execution performance and the average effectiveness. We then introduce an equi-number checkpointing strategy. We derive the metrics for expected program execution time and average effectiveness under some assumptions. We show the influence of checkpointing and handoff to these two metrics, and determine the conditions when checkpointing is beneficial. In addition, we derive the optimal checkpointing interval for minimizing the total program execution time and maximizing the average effectiveness. Finally we show the results of simulations and the comparisons between the performance and effectiveness of programs with and without checkpointing and draw our conclusions. Note throughout the paper we use the word "failure" to mean component or program failure. It could be termed as "fault" as it does not represent system failure.

2. Related Work

Many researchers have studied the problem of optimizing the checkpointing interval that reduces the overhead caused by rollbacks. Young [23] presents a first-order approximation to the optimal time interval between checkpoints. He assumes that (1) a failure is detected as soon as it occurs; (2) the checkpointing interval is fixed; (3) the inter-failure time is exponentially distributed; (4) the checkpointing time is constant; and (5) no failures occur during error recovery [20]. In our approach, we also assume instant failure detection but the checkpointing time is a random variable with general distribution. Tantawi and Ruschitzka [20] consider general failure distributions and allow failures to occur during checkpointing and error recovery. They introduce an equi-cost strategy which is a failure-dependent yet reprocessing-independent checkpointing strategy. This paper will only consider that the inter-failure time is exponentially distributed. However, we improve over [20] by taking general handoff distributions and failures during checkpointing and handoff into consideration besides handoffs during checkpointing and rollback. Duda [7] derives the distribution and expectation of program execution time with and without checkpointing. Nicola [16] analyzes program execution time with different checkpointing strategies: equi-distant checkpointing, checkpointing in modular programs, and random checkpointing. In [7] and [16], they point out that the expected elapse time of programs without checkpointing increases as an exponential function of the execution time, whereas the use of checkpointing causes a linear increase. Chandy [3] discusses costs of rollback and recovery with several assumptions: the checkpointing time is fixed, the time required to reprocess the logged transactions is proportional to the number of transactions recorded since the last checkpoint, and the checkpoint itself is always correct. In [4], Chandy et al. propose algorithms which make use of estimates made by the programmers on the maximum amount of processing time to minimize the maximum or expected time spent in checkpointing. These algorithms need some *a priori* knowledge about the processing time. In [12], Krishna et al. point out that the optimization criteria for checkpoint placement is to trade the benefits derived from checkpointing with the overhead imposed. Ziv and Bruck [24] present an online algorithm for placement of checkpoints. The algorithm keeps track of the state size of a program, and looks for points in the program where checkpoint placement is the most beneficiary. Gelenbe [10] utilizes the queueing process related to the requests for transaction processing arriving at a database system and shows that the optimal checkpointing interval is a function of the load of the database system. We will utilize the received computational message number for checkpoint placement and derive an approximation to the optimal message number interval between checkpoints. Tsai et al. [22] prove that there is no optimal checkpointing strategy with rollback-dependency trackability for all possible communication patterns. We will also point out informally that different checkpointing strategies should be employed for suiting different computing environments.

Neves and Fuchs [15] propose a coordinated checkpoint protocol for distributed systems in mobile environments. Checkpoints are saved remotely on stable storage, or locally in MHs. Chen and Lyu [5] present a message logging and recovery strategy in Wireless CORBA, in which checkpoints are also saved remotely on stable storage. In this paper, we also assume that checkpoints are saved remotely. Pradhan et al. [19] discuss the design and trade-off in recoverable mobile environments. They choose the Mobile Support Station (MSS) as stable storage for an MH. They identify the trade-off parameters for recovery strategy, which include failure rate of an MH, communication/mobility ratio, wireless network factor, etc. They analyze the costs of the proposed recovery schemes, but they do not analyze the expected program execution time and effectiveness. We will discuss how the checkpointing and handoff rate influence these two metrics.

3. Mobile Architecture

Distributed mobile computing environments have been addressed in hardware entities by many papers in the literature [1, 15]. In mobile distributed computing, much of the action takes place in the middleware level. Therefore, we regard that describing the mobile architecture using middleware entities would be more suitable. We borrow a mobile architecture from Telecom Wireless CORBA (Common Object Request Broker Architecture) [17], which is standardized by Object Management Group (OMG). This architecture is shown in Figure 1.

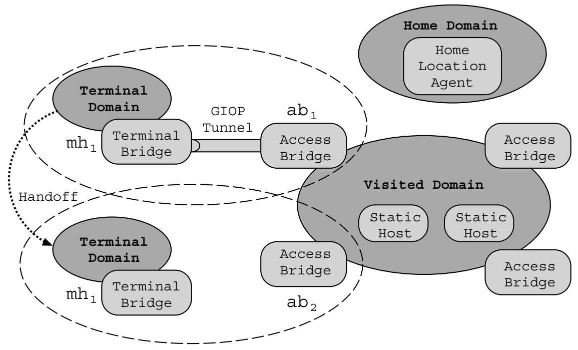

Figure 1. Wireless CORBA architecture

In Figure 1, wireless CORBA identifies three 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 other objects in wired or wireless networks. The Visited Domain contains several Access Bridges (ABs) to provide communications with objects on MHs. It also contains some static hosts. All communications in the Visited Domain are via wired links. An MSS contains necessary wireless facilities to communicate with MHs and includes wired interfaces to communicate with static hosts and other MSSs. The Home Domain is composed of the Home Location Agent which keeps track of the ABs that the MH has associated with when it moves around.

Each AB covers a geographical area within which it can communicate with the corresponding MHs directly, which is plotted as dashed circle in Figure 1. When an MH moves across the border of a 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 the handoff, no computational messages can be transmitted between the ABs and the MH.

4. Program Execution without Checkpointing

As mentioned before, GTP messages are transmitted through GIOP tunnels via wireless links during a program execution. These messages can be classified into two categories: 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 vary as failures and handoffs, when occurring, are random events. We consider the execution time of a program in an MH with a given number of computational messages in the presence of random failures and handoffs.

4.1. Model Description

Let us first introduce the notations that will be used in this paper. Let Z, a random variable, be the continuously processing time requirement of a task in the absence of failures, handoffs and checkpointings. During a task execution, three events may occur: failures, handoffs and checkpointings, denoted as f, h, and c respectively. We define that $Z^{(e)}$ represents the task execution time in the presence of event e, $e = f, h, c$, with the time requirement Z. Multiple event types may occur during task execution, e.g., $Z^{(f,h)}$ is the task execution time in the presence of failures and handoffs. The general cumulative distribution function (c.d.f.) of the random variable Z is $G_Z(t)$. We denote the Laplace-Stieltjes Transform (LST) of the c.d.f. of Z as $\phi_Z(s) = \int_{t=0}^{\infty} e^{-st} dG_Z(t) = E(e^{-sZ})$. Let $\tilde{Z}$ be a random variable which has the same probability distribution with Z, then $\phi_{\tilde{Z}}(s) = \phi_Z(s)$.

Now we turn to describe the program execution model in the absence of checkpointing. Let n denote the number of computational messages that a program in an MH should receive. The messages arrive according to a Poisson process at rate λ. $X(n)$ is the program execution time with the required computational message number n. The instants of the occurrences of failures form a homogeneous Poisson process of parameter γ and time between two successive handoff events is modelled as an exponential distribution with parameter ρ. The program has three states during its execution, denoted as State 0, 1, and 2 shown in Figure 2[1], respectively. State 0 is the normal (operational) state, in which messages can be received. If a handoff occurs, the program transits to State 2. The handoff time H is a random variable with general distribution. The program will enter State 1 if a failure occurs when the program is in State 0 or 2. A repair process will be conducted instantly when entering State 1. The repair time requirement R is also regarded as a random variable with general distribution. The

[1]In state transition diagrams, we only show the transition rates which have been given explicitly in our assumption.

restarting time is assumed to be much smaller than R, so we ignore it here. During State 1, failures may still occur, after which the repair is repeated [13], but handoffs cannot be made as the MH does not work in this state. Eventually the program returns to State 0. We assume a failure is detected as soon as it occurs.

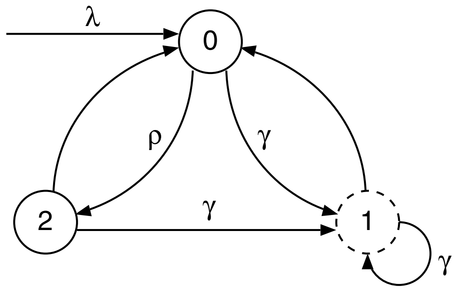

Figure 2. State transition in program execution without checkpointing

Without checkpointing, the program should be restarted from its beginning after a failure, as all the computation from the beginning to the failure epoch is lost. This computation is denoted as wasted computation [18]. The program will terminate successfully if it receives n messages continuously before the next failure. During a handoff, no computational message is distributed to the MH. Only after the completion of a handoff, computational messages can be transmitted continuously. Without loss of generality, we assume that the program starts at State 0.

4.2. Execution Time and Average Effectiveness without Checkpointing

If a program enters State 1, it must complete a repair task with the time requirement R, after which it returns to State 0. We may calculate the sojourn time in State 1 independently. So the State 1 is plotted with dashed circle in Figure 2.

Lemma 1. The LST of the c.d.f. of $R^{(f)}$, the repair time in the presence of failures with the time requirement R, i.e., the sojourn time in State 1, is given by

$$\phi_{R^{(f)}}(s) = \frac{(s+\gamma)\phi_R(s+\gamma)}{s+\gamma\phi_R(s+\gamma)}, \tag{1}$$

and the expectation of $R^{(f)}$ is

$$E(R^{(f)}) = \frac{1-\phi_R(\gamma)}{\gamma\phi_R(\gamma)}. \tag{2}$$

Proof[2]*:* Let Y be the time to the first failure after starting repair, then we have

$$R^{(f)} = \begin{cases} R & : \ if \ \ R < Y \\ Y + \tilde{R}^{(f)} & : \ if \ \ R \geq Y. \end{cases}$$

If $R < Y$, then a repair will be successful completed without failures, so the repair time is R. If $R \geq Y$, a failure occurs after which another repair is simply repeated, denoted as $\tilde{R}^{(f)}$. Thus the repair time is $Y + \tilde{R}^{(f)}$ in this case. Taking conditional expectation of $R^{(f)}$, we get

$$E(e^{-sR^{(f)}}|R,Y) = \begin{cases} e^{-sR} & : \ if \ \ R < Y \\ e^{-sY}E(e^{-s\tilde{R}^{(f)}}) & : \ if \ \ R \geq Y, \end{cases}$$

as Y should be independent of $\tilde{R}^{(f)}$. Unconditioning on Y, we have

$$\begin{aligned} & E(e^{-sR^{(f)}}|R) \\ = & \int_{y=0}^{\infty} E(e^{-sR^{(f)}}|R, Y=y) \cdot \gamma e^{-\gamma y}\, dy \\ = & \ e^{-(s+\gamma)R} + \frac{\gamma E(e^{-s\tilde{R}^{(f)}})}{s+\gamma}\left[1 - e^{-(s+\gamma)R}\right]. \end{aligned}$$

Removing the condition on R, the result is

$$\phi_{R^{(f)}}(s) = \phi_R(s+\gamma) + \frac{\gamma\phi_{R^{(f)}}(s)}{s+\gamma}\left[1 - \phi_R(s+\gamma)\right]. \tag{3}$$

Rearranging the above equation yields Equation (1). After engaging the moment generating property of the Laplace transform [6], $E(R^{(f)}) = -d\phi_{R^{(f)}}(s)/ds|_{s=0}$, the expected repair time in the presence of failures is given by Equation (2). ■

Theorem 1. Let

$$Q(s) = \frac{\lambda}{s+\lambda+\gamma+\rho-\rho\phi_H(s+\gamma)} \tag{4}$$

and

$$q = \frac{\lambda+\gamma+\rho-\rho\phi_H(\gamma)}{\lambda}, \tag{5}$$

then the LST of the c.d.f. of the program execution time $X^{(f,h)}(n)$ contains the form

$$\phi_{X^{(f,h)}}(s,n) = \frac{(s+\gamma)Q^n(s)}{s+\gamma-\gamma\phi_{R^{(f)}}(s)[1-Q^n(s)]}, \tag{6}$$

and the expectation of $X^{(f,h)}(n)$ is

$$E(X^{(f,h)}(n)) = \left[\frac{1}{\gamma} + E(R^{(f)})\right](q^n - 1). \tag{7}$$

Proof: Under the assumptions stated in Section 4.1, the random variable $X(n)$ inherits an N-stage Erlang distribution with parameter λ [21]. Let K, $K = 0, 1, 2, \ldots$, be the number of transitions from State 0 to State 2, i.e., the number

[2]The proof follows similar approaches in [9, 16].

of handoffs during a normal execution. So the total time requirement is $X(n) + \sum_{j=1}^{K} H_j$. Let Y be the time to the first failure event after starting program execution. Then we have

$$X^{(f,h)}(n) = \begin{cases} X(n) + \sum_{j=1}^{K} H_j \\ \quad : \quad if \;\; X(n) + \sum_{j=1}^{K} H_j \le Y \\ Y + R^{(f)} + \tilde{X}^{(f,h)}(n) \\ \quad : \quad if \;\; X(n) + \sum_{j=1}^{K} H_j > Y. \end{cases}$$

If $X(n) + \sum_{j=1}^{K} H_j \le Y$, the program will make K handoffs before it receives n messages without failures. In this case, the total handoff time is $\sum_{j=1}^{K} H_j$ and the total program execution time is $X(n) + \sum_{j=1}^{K} H_j$. $H_j, j = 1, \ldots, K$, are independent and identically distributed (i.i.d.) random variables. If $X(n)+\sum_{j=1}^{K} H_j > y$, a failure occurs before the program receives n messages and makes K handoffs. In this case, there is a repair time $R^{(f)}$ after which the program execution is restarted from its beginning, which means that the program is required to receive n messages without failure interruptions again [16]. Following similar steps in the proof of Lemma 1, the theorem is proofed. ■

Equation (7) shows that the expectation of program execution time is an exponential function of the number of messages n. It also depends on failure rate γ, message arrival rate λ and handoff rate ρ. In Equation (7), $[1/\gamma + E(R^{(f)})]$ corresponds to the expectation of inter-failure time and the following repair time. Moreover, the second term, $(q^n - 1)$, denotes the expected number of failure occurrences when the program is in State 0 and 2.

Remark 1. With no failures during program execution, i.e., γ tends to 0, then

$$\lim_{\gamma \to 0} E(X^{(f,h)}(n)) \quad = \quad \frac{n}{\lambda}\left[1 + \rho E(H)\right]. \qquad (8)$$

Average effectiveness can be defined as how much of the time an MH performs effective computation towards its completion during an execution. An execution of a program in an MH may be decomposed into wasted and effective intervals [18]. If a program produces useful work towards its completion then it is in *effective interval*, while a *wasted interval* is one in which the program does not produce useful work. The times when the MH is in repair or in handoff are considered as wasted times. If the MH is functional but the program is not producing useful computation, as mentioned as wasted computation before, the time interval is also regarded as a wasted interval. According to these definitions of effective interval and wasted interval, the effective time during an execution can be calculated by the expected execution time of a program without failures and handoffs, which is n/λ, as the program has to receive n messages successfully. The sum of the wasted time and effective time, on the other hand, can be computed by the expected execution time of a program in the presence of failures and handoffs, which is $E(X^{(f,h)}(n))$. Consequently the average effectiveness of a program execution without checkpointing, denoted as $A(n)$, is

$$A(n) = \frac{n}{\lambda \cdot E(X^{(f,h)}(n))}. \qquad (9)$$

$A(n)$ decreases as message number n or handoff arrival rate ρ increases and as message arrival rate λ decreases. From the above analysis, we know that most execution time is wasted due to failures. So we need some techniques to cut down the wasted computation time and increase the average effectiveness. Checkpointing and rollback are such techniques suitable to achieve this object.

5. Equi-number Checkpointing

Equi-number checkpointing means checkpoints are equally placed with respect to the number of received messages n. It can be treated in two ways: one is to fix the message number in each checkpointing interval, and the other is to fix the total checkpoint interval number. We denote both of them as equi-number checkpointing. The former is with respect to message number, and the latter is with respect to checkpoint number. The first one is very naturally derived from the equi-distant checkpointing strategy [16] as we change the required program execution time to the required total message number.

5.1. Model Description

The number of messages n is divided into m equal intervals (supposing n could be divided exactly by m), so each interval has $a = n/m$ messages. Despite of n, the checkpointing strategy is equi-number checkpointing with respect to checkpoint number if m is fixed, and it is equi-number checkpointing with respect to message number if a is fixed. As these two approaches have the same mathematical expression of the expected execution time and average effectiveness, we will only consider equi-number checkpointing with respect to message number. For simplicity, when a program finishes we also take a checkpoint, so there is always a checkpoint at the end of each interval, forming a total of m checkpoints. The execution times of the m intervals, $X_i(a), i = 1, 2, \ldots, m$, are assumed as i.i.d. random variables. In each interval, the execution is the same as the execution without checkpointing except that the program restarts from its most recent checkpoint.

Figure 3(a) shows the state transition during execution with equi-number checkpointing in the $i^{th}, i = 1, 2, \ldots, n$, interval. There are two composite states 3 and 4, whose detailed representations are showed in Figure 3(b) and 3(c). State 3 corresponds to the composite repair and rollback

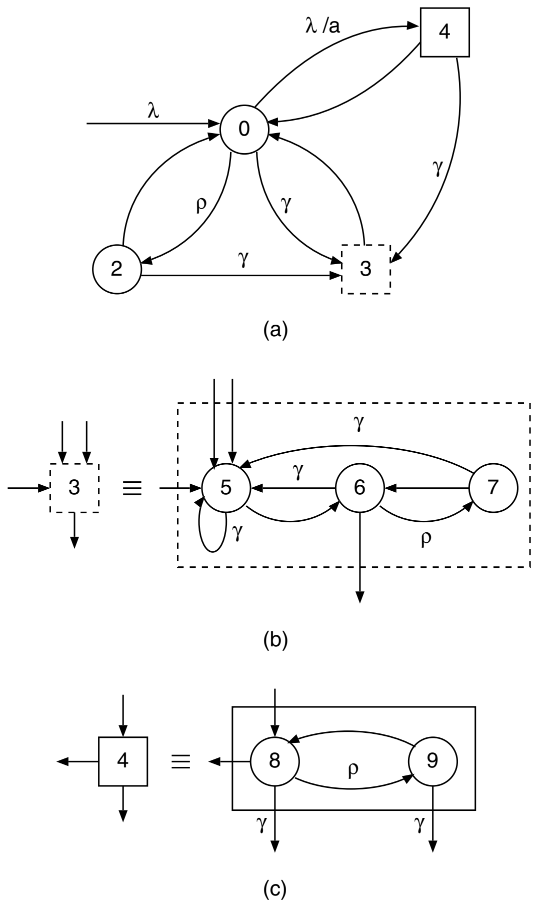

Figure 3. State transition in program execution with equi-number checkpointing

state, in which State 5, 6, and 7 denote repair, rollback, and handoff, respectively. After undergoing a successful repair process in State 5, the program enters rollback state instantly. State 4 is the composite checkpoint creation state, in which State 8 denotes the checkpoint creation state and State 9 denotes the handoff state during checkpointing. Let C, a general distributed random variable, denote the checkpoint creation time. Therefore C is the execution time requirement in State 8. We know C contains two parts of time: the time to take a checkpoint and the time to save the checkpoint on stable storage. In our mobile network model [5], the MH is not safe to be treated as stable storage. Checkpoints should be transmitted through a wireless link, which may suffer low bandwidth and long transfer delay, before being saved on the stable storage of the currently associated AB. Therefore, the checkpointing saving part is the more significant one. After a failure, a recent checkpoint should be reloaded into the MH through a wireless link. For the same reason, the main part of the rollback time is consumed by checkpoint transmission. So we let the rollback time also be C, which follows that the sojourn time in State 3 is $[R + C^{(h)}]^{(f)}$, denoted as R'.

5.2. Execution Time and Average Effectiveness with Checkpointing

The dashed rectangle with State 3 in Figure 3 denotes that a program must complete a repair and rollback task before it enters State 0. We first calculate the sojourn time in State 3.

Lemma 2. Let

$$B(s) = \phi_C[s + \gamma + \rho - \rho\phi_H(s + \gamma)], \qquad (10)$$

then the LST of the c.d.f. of R', the sojourn time is State 3, is given by

$$\phi_{R'}(s) = \frac{(s + \gamma)\phi_R(s + \gamma)B(s)}{s + \gamma\phi_R(s + \gamma)B(s)}, \qquad (11)$$

and the expectation of $E(R')$ is

$$E(R') = \frac{1 - \phi_R(\gamma)\phi_C[\gamma + \rho - \rho\phi_H(\gamma)]}{\gamma\phi_R(\gamma)\phi_C[\gamma + \rho - \rho\phi_H(\gamma)]}. \qquad (12)$$

Proof: The time requirement of repair and rollback in the presence of failures and handoffs is $R + C + \sum_{j=1}^{K} H_j$, where K, $K = 0, 1, 2, \ldots$, is the number of occurrences of handoffs during rollback. Note that the checkpoint creation time C is a random variable; however, it is fixed for a given repair and rollback process [16]. Following similar steps in Lemma 1 we can obtain the Equation (11) and (12). ■

After getting the sojourn time in State 3, we now analyze the program execution with checkpointing in the presence of failures and handoffs.

Theorem 2. Let $X(n, a)$ denote the total execution time of a program which is divided into n/a (assumed as an integer) checkpointing intervals, and each interval should receive a messages continuously without failure. The LST of the c.d.f. of $X^{(c,f,h)}(n, a)$ is given by

$$\begin{aligned}&\phi_{X^{(c,f,h)}}(s, n, a)\\ =\;& \left[\frac{(s + \gamma)B(s)Q^a(s)}{s + \gamma - \gamma\phi_{R'}(s)\left[1 - B(s)Q^a(s)\right]}\right]^{\frac{n}{a}}, \quad (13)\end{aligned}$$

and the expectation of program execution time is

$$\begin{aligned}&E(X^{(c,f,h)}(n, a))\\ =\;& \frac{n}{a}\left[\frac{1}{\gamma} + E(R')\right]\left[\frac{q^a}{\phi_C(\gamma + \rho - \rho\phi_H(\gamma))} - 1\right] (14)\end{aligned}$$

Proof: First we only consider the program execution time in the i^{th} interval. K_i and Y_i are the number of transitions from State 0 to State 2 and the time to the first failure in the i^{th} interval, respectively. Then we have

$$X_i^{(c,f,h)}(a) = \begin{cases} X_i(a) + \sum_{j=1}^{K_i} H_j + C \\ \quad : \quad if \;\; X_i(a) + \sum_{j=1}^{K_i} H_j + C \le Y_i \\ Y_i + R' + \tilde{X}_i^{(c,f,h)}(a) \\ \quad : \quad if \;\; X_i(a) + \sum_{j=1}^{K_i} H_j + C > Y_i. \end{cases}$$

Following similar steps, we have $\phi_{X_i^{(c,f,h)}}(s,a)$ and $E(X_i^{(c,f,h)}(a))$. The total execution time $X^{(c,f,h)}(n,a)$ is the sum of n/a i.i.d. random variables $X_i^{(c,f,h)}(a)$, so the LST of $X^{(c,f,h)}(n,a)$ is

$$\phi_{X^{(c,f,h)}}(s,n,a) = [\phi_{X_i^{(c,f,h)}}(s,a)]^{\frac{n}{a}},$$

and

$$E(X^{(c,f,h)}(n,a)) = \frac{n}{a} E(X_i^{(c,f,h)}(a)).$$

Finally, we obtain Equation (13) and (14). ■

Equation (14) shows that $E(X^{(c,f,h)}(n,a))$ is a linear function of the number of messages n and an exponential function of the number of messages a in each checkpointing interval. In Equation (14), n/a corresponds to the number of checkpoints during an execution. The other term has a similar expression with Equation (7) except that it has one more factor $1/\phi_C(\gamma + \rho - \rho\phi_H(\gamma))$ which is contributed by the fact that the time requirement in each interval adds a checkpoint creation time. Note that with checkpointing, the expectation of program execution time decreases dramatically: in Equation (7) it is an exponential relationship with n, while in Equation (14) it reduces to a linear relationship with n.

Remark 2. With no failures during program execution, i.e., γ tends to 0, then

$$\lim_{\gamma \to 0} E(X^{(c,f,h)}(n,a)) = \frac{n}{a}\left[\frac{a}{\lambda} + E(C)\right][1 + \rho E(H)]. \tag{15}$$

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 wasted time, then the average effectiveness $A(n,a)$ can be computed from

$$A(n,a) = \frac{n}{\lambda \cdot E(X^{(c,f,h)}(n,a))}, \tag{16}$$

which has the same form as Equation (9). With equi-number checkpointing with respect to message number, $A(n,a)$ is a constant and does not vary with n. But it will vary with n in equi-number checkpointing with respect to checkpointing number, which has the same curve shape as $A(n)$.

We know that the benefit of checkpointing comes with a price. During failure-free execution, checkpointing delays the message delivery. After a failure, time is required to reload the program status stored in a checkpoint. There usually exists an optimal checkpointing interval which minimizes the total program execution time or maximizes the average effectiveness. Numerically, $E(X^{(c,f,h)}(n,a))$ or $A(n,a)$ are observed to be convex functions of a, although it is difficult to prove them analytically. So the optimal checkpointing interval $\hat{a}$ can be obtained by solving $(\partial/\partial a)[E(X^{(c,f,h)}(n,a))] = 0$ or $(\partial/\partial a)[A(n,a)] = 0$. The derived equation is

$$q^a\,[1 - a\ln q] = \phi_C(\gamma + \rho - \rho\phi_H(\gamma)). \tag{17}$$

Using the Maclaurin expansion of q^a as far as the second degree term, the approximate solution of the Equation (17) is

$$\hat{a} \cong \left\lfloor \frac{\sqrt{2[1 - \phi_C(\gamma + \rho - \rho\phi_H(\gamma))]}}{\ln q} \right\rfloor, \tag{18}$$

from which we know that $\hat{a}$ is independent of the required message number n, but it depends not only on the failure rate γ, the message arrival rate λ and the handoff rate ρ, but also on the distribution of checkpoint duration and handoff duration. The optimal checkpoint count is $\hat{m} = \lfloor n/\hat{a} \rfloor$. We denote the minimal value of the expected program execution time and the maximal value of the average effectiveness as $E^*(X^{(c,f,h)}(n,a))$ 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 $E(X^{(c,f,h)}(n,a)) < E(X^{(f,h)}(n))$, which can be approximated by

$$n > 2\left\lfloor \frac{\frac{q^a}{\phi_C(\gamma+\rho-\rho\phi_H(\gamma))} - 1}{a\,(\ln q)^2} - \frac{1}{\ln q} \right\rfloor. \tag{19}$$

6. Simulations and Comparisons

In this section, we will conduct simulations to verify our analytical results and evaluate the effects of parameters' selection on average effectiveness.

6.1. Simulation Results

To verify the correctness of the expected execution time with and without checkpointing we derived, we perform two software simulations. In these two simulations, we consider the following parameter values:

- mean message arrival rate $\lambda = 10^{-2}$ and mean handoff rate $\rho = 10^{-3}$;
- mean failure arrival rate $\gamma = 10^{-3}, 10^{-4}, 10^{-5}$;
- checkpoint creation time C, repair time R, and handoff time H are exponentially distributed and $E(C) = E(R) = 20, E(H) = 10$;

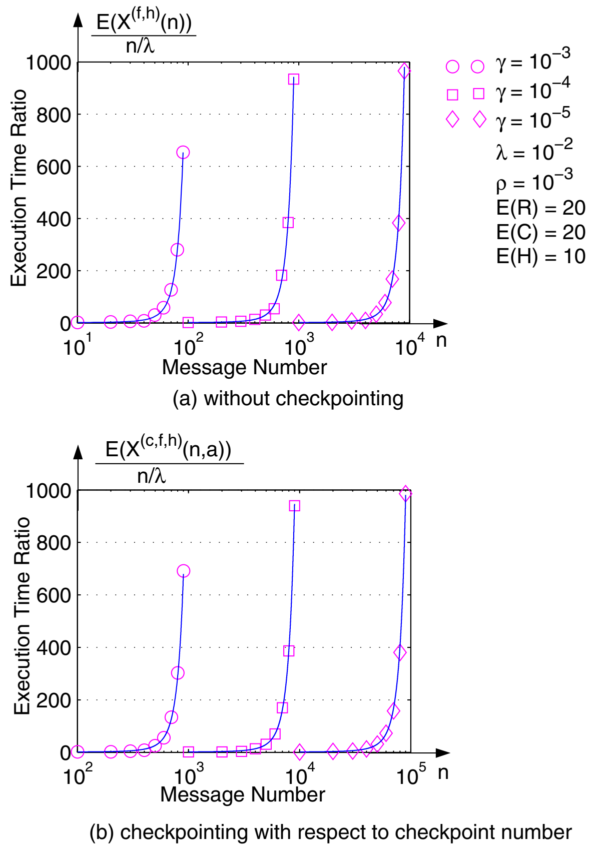

Figure 4. Simulation result without checkpointing and with checkpointing

- checkpoint count m is fixed and is 10.

We run each simulation 50 times with different failure arrival rates and the results of the mean execution time are shown in Figure 4[3]. These two figures show that the simulations (denoted by discrete marks in the figures) 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 failures to the expectation without failures. In each case, the curves increase exponentially and have the similar shapes as the number of required messages increases. Figure 4(a) and 4(b) are similar, except for the labels of X axis. The difference between them is the number of checkpointing intervals m. We know that in each checkpointing interval, the execution process is similar with the process without checkpointing. From another point of view, checkpointing decreases the failure arrival rate to $1/m$ of the failure arrival rate without checkpointing .

[3]The expected execution time and the average effectiveness are discrete functions of the message number n, but for clarity we plot the figures in continuous curves.

6.2. Analytical Results Comparisons

Figure 5 shows the result of average effectiveness as we change the value of checkpoint count m. The extremal situation is that there is a checkpoint after each received message. Therefore with equi-number checkpointing with respect to checkpoint number, n should start with m, as shown in Figure 5. From this figure, we see that for the given message arrival rate λ, failure arrival rate γ, and handoff rate ρ, there is a descend of average effectiveness when the required message count decreases to a certain value. There also are intersections between curves representing different checkpoint counts, which means more checkpoints may bring more overhead when the message number is low. Certainly, as the required message number increases, more checkpoints will be more beneficial. The dashed curve represents the average effectiveness without checkpointing, which decreases monotonically as the required message number increases. The less the message number is, the higher the effectiveness is. Eventually it achieves higher effectiveness than taking checkpoints during execution. This confirms the general knowledge that checkpointing is not always beneficial to the program execution time or the average effectiveness. We will also see this effect in Figure 6. The curve with $m = 1$ is always below the dashed curve for with $m = 1$ a program has a checkpoint creation period additionally.

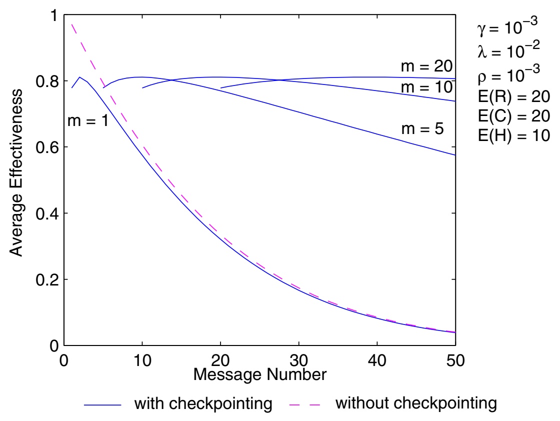

Figure 5. Equi-number checkpointing with respect to checkpoint number

To illustrate whether the checkpointing can be used to reduce the overall execution time or increase the average effectiveness, we compare the average effectiveness with and without checkpointing from another viewpoint. This time we utilize equi-number checkpointing with respect to

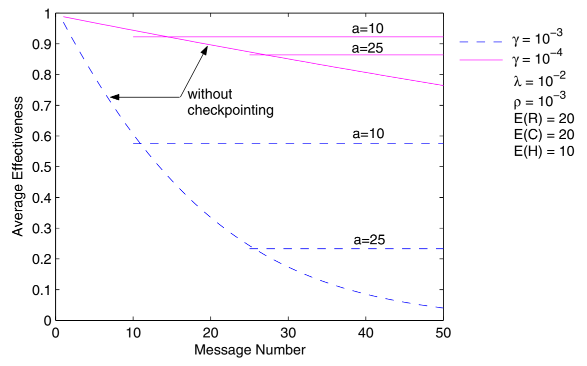

Figure 6. Equi-number checkpointing with respect to message number

message number, i.e., the number of required messages a is a constant in each checkpointing interval, under which the average effectiveness is a constant, shown as horizontal lines in Figure 6. The average effectiveness increases as a decreases, which means more checkpoints are to be taken in execution. However, excessive checkpointing may decrease the effectiveness, as shown later in Figure 7. These horizontal lines have intersections with those curves which represent no checkpoint is taken during execution. If the required program message number is below these intersections, a program can run more economically by not taking checkpoints and by rerunning from the beginning of the program [11].

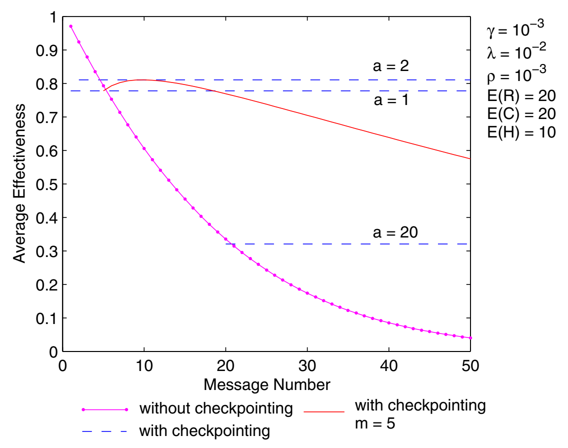

Figure 7. Comparison between checkpointing and without checkpointing

Figure 7 shows that when we engage equi-number checkpointing, there exists an optimized $\hat{a}$ maximizing the average effectiveness. The approximate solution of $\hat{a}$ is expressed in Equation (18). Under the parameters provided in this figure, $\hat{a} = 2$ and $A^*(n, a) = 0.811$. With equi-number checkpointing with respect to checkpoint number m and $m = 5$, the effectiveness gets maximal value as $n = 10$. For $n < 6$, checkpointing cannot gain any benefit on the effectiveness.

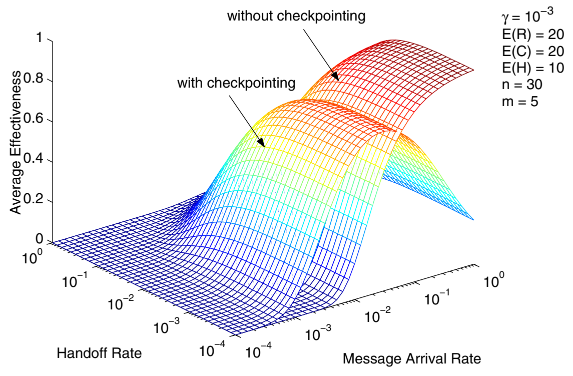

Figure 8. Average effectiveness vs. message arrival rate and handoff rate

Figure 8 demonstrates the variation of average effectiveness with message arrival rate λ and handoff rate ρ when without checkpointing or engaging checkpointing with respect to checkpoint number. The average effectiveness decreases as ρ increases, despite engaging checkpointing or not. The effectiveness increases as λ increases when the program executes without checkpointing. But for checkpointing with respect to checkpoint number, the effectiveness increases first and then decreases. When the message arrival rate is low, checkpointing increases the effectiveness. But when the message arrival rate is high, the program will be completed by experiencing less failures, and most of the checkpoints do not contribute to the effectiveness. Consequently, checkpointing incurs more overheads. Under this condition, we should take checkpoints less frequently to reduce these overheads.

Figure 5– 8 demonstrate that there are several factors to be weighted before determining whether and how checkpoint is worth taking [11]. These factors include failure arrival rate, message arrival rate, handoff rate, checkpoint creation cost, message number, etc. Besides those factors mentioned above, other factors, such as power consumption, and so on, should also be evaluated. In summary, if we can adaptively adjust the checkpointing parameters and choose among various checkpointing strategies, we will get more performance and effectiveness improvement.

7. Conclusions

In this paper, we perform the analysis of program execution time and average effectiveness with and without checkpointing in mobile environments, under the assumption that the number of totally received computational messages n is not changed during a required program execution, and the inter-failure arrival time, the inter-message arrival time and the inter-handoff time are exponentially distributed. We consider a general mobile network architecture, and describe an equi-number checkpointing strategy in this architecture. Analytical results show that the program execution time is an exponential function of n without checkpointing and is a linear function of n with checkpointing. We confirm our analytical results with simulations. We also demonstrate the trade-offs between checkpointing versus non-checkpointing, and compare them quantitatively in different conditions. The impact of various parameters in mobile network environments is evaluated regarding whether and how checkpointing should be taken for program performance and effectiveness improvements.

Acknowledgement

The work described in this paper was fully supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4360/02E).

References

[1] A. Acharya and B. R. Badrinath. Checkpointing distributed applications on mobile computers. *The 3rd International Conference on Parallel and Distributed Information Systems*, pages 73–80, September 1994.

[2] R. Bagrodia, W. W. Chu, L. Kleinrock, and G. Popek. Vision, issues, and architecture for nomadic computing. *IEEE Personal Communications*, 2(6):14 –27, December 1995.

[3] K. M. Chandy. A survey of analytic models of rollback and recovery strategies. *Computer*, 8(5):40–47, May 1975.

[4] K. M. Chandy and C. V. Ramamoorthy. Rollback and recovery strategies for computer programs. *IEEE Transactions on Computers*, 21(6):546–556, June 1972.

[5] X. Chen and M. R. Lyu. Message logging and recovery in wireless CORBA using access bridge. *The 6th International Symposium on Autonomous Decentralized Systems*, pages 107–114, April 2003.

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

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

[8] 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.

[9] D. P. Gaver Jr. A waiting line with interrupted service, including priorities. *Journal of the Royal Statistical Society, Series B*, 24:73–90, 1962.

[10] E. Gelenbe. On the optimum checkpoint interval. *Journal of ACM*, 26(2):259–270, April 1979.

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

[12] C. M. Krishna, K. G. Shin, and Y.-H. Lee. Optimization criteria for checkpoint placement. *Communications of the ACM*, 27(10):1008–1012, October 1984.

[13] V. G. Kulkarni, V. F. Nicola, and K. S. Trivedi. Effects of checkpointing and queueing on program performance. *Communications in Statistics - Stochastic Models*, 6(4):615–648, 1990.

[14] 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.

[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. Nicola. 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] 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.

[19] 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.

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

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

[22] J. Tsai, S.-Y. Kuo, and Y.-M. Wang. Theoretical analysis for communication-induced checkpointing protocols with rollback-dependency trackability. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):963–971, October 1998.

[23] J. W. Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 16(9):530–531, September 1974.

[24] A. Ziv and J. Bruck. An on-line algorithm for checkpoint placement. *IEEE Transactions on Computers*, 46(9):976–985, September 1997.