Checkpoint Placement Algorithms for Mobile Agent System

Jin Yang, Jiannong Cao, Weigang Wu
Internet and Mobile Computing Lab
Department of Computing, Hong Kong Polytechnic University
Hung Hom, Kowloon Hong Kong
{csyangj, csjcao, cswgwu}@comp.polyu.edu.hk

Abstract

Checkpointing is a fault tolerance technique widely used in various types of computer systems. In checkpointing, an important issue is how to achieve a good trade-off between the recovery cost and the system performance. Excessive checkpointing would result in the performance degradation due to the high costly I/O operations during checkpointing. Equidistant and equicost are two well-known checkpointing strategies for addressing this issue. However, there is no study on these strategies catering for a mobile agent (MA) system, which has different characteristics with conventional systems. In this paper, based on an analysis of the behaviours of an MA system, we find that it can be modelled as a homogeneous discrete-parameter Markov chain, which is different from the models used in conventional systems. Therefore, the analytic methods and corresponding results for conventional systems cannot be adopted directly for an MA system. Based on our proposed model, we study the equidistant and equicost checkpointing strategies and propose checkpoint placement algorithms for MA systems. Through simulations we evaluate the performance of our proposed algorithms and the result shows that the equicost strategy based algorithm is most suitable for an MA system.

1. Introduction

A mobile agent (MA) is a program that can migrate from host to host in a network of heterogeneous computer systems to execute the tasks specified by its owner. Characteristics of MAs include mobility, autonomy, asynchrony, encapsulation of protocols, adaptability, support for mobile computing (disconnected operations), etc. Among these characteristics, autonomy is an attractive feature which allows an MA to determine its itinerary dynamically. As a result, an MA can have a self-initiated itinerary which is more flexible than pre-defined itinerary. Combining the characteristics of distributed computing and mobile computing, MA has been used for structuring and coordinating distributed applications [1, 2, 3]. Many of these applications require high degree of reliability, such as electronic commerce, network management, and information collection and fusion. Therefore, fault tolerance is a key issue in designing an MA system.

Checkpointing is a technique to achieve fault tolerance and has been widely used in various kinds of computer systems. It is also proposed for MA systems, where a checkpoint is the copy of an MA's state, including the partial result obtained and the execution status, which can be used during recovery to resume the execution of the MA. Normally, an MA system periodically saves checkpoints of the MAs and stores them on stable storage. When the failure of an MA is detected by the system, it will recover the MA by rolling back to its last checkpoint. Checkpointing can be naturally done in an MA system: serializing an MA for the migration to the next host effectively constructs a checkpoint. This is especially true for most MA systems developed in Java. Various kinds of checkpointing techniques have been developed [4], including independent (or uncoordinated), coordinated, or communication-induced schemes. Independent checkpointing is the simplest scheme, which allows an MA to take checkpoints periodically without any coordination with other MAs. Independent checkpointing has been used to store the persistent state of a single MA and guarantee the reliable migration of MAs [9, 12, 18].

Checkpointing involves high costly I/O operations, so how to achieve a good trade-off between the checkpointing cost and system performance is a critical issue. Excessive checkpointing would result in the performance degradation during normal operation. On the contrary, deficient checkpointing would incur expensive recovery cost upon a failure. There has been

Proceedings of the Fifth International Conference on Grid and Cooperative Computing (GCC'06)
0-7695-2694-2/06 $20.00 © 2006 IEEE

much research on how to determine the optimal checkpointing interval, which is referred to as the checkpoint placement problem if the optimal checkpointing interval cannot be achieved. Equidistant and equicost are the two well-known checkpointing strategies. The equidistant strategy considers a deterministic productive time between two neighboring checkpoints, while the equicost strategy allows a checkpoint to be made when the expected re-execution cost is equal to the checkpointing cost. With the occurrence of failures following a Poisson process, these strategies become identical and will result in the optimal checkpointing interval [16] in conventional systems.

Currently, for mobile agent systems, there is no work done on how to determine a proper checkpointing interval for an MA and no study on how the above two strategies can be applied. In this paper, we firstly analyze the behaviours of an MA system. Based on the analysis, we find that it can be modeled as a homogeneous discrete-parameter Markov chain, which is different from the models used in conventional systems. Therefore, the analytic methods and corresponding results for conventional systems cannot be adopted directly for an MA system. Based on our proposed model, we study the equidistant and equicost checkpointing strategies and propose three checkpoint placement algorithms for MA systems. Through simulations we evaluate the performance of our proposed algorithms and the result shows that the equicost strategy based algorithm is most suitable for an MA system.

The rest of the paper is organized as follows. Section II briefly describes previous works. Section III defines our system model. A scheme to determining the optimal checkpointing interval for an MA under ideal condition and algorithms for checkpoint placement for an MA under general condition are proposed in Section IV. Simulation results are presented in Section V. We conclude this paper in Section VI.

2. Related works

Determining the optimal checkpointing interval has been studied for a long time. Most works focus on the uniprocessor systems [6, 7, 8, 10, 15, 16, 19]. They use execution time as the basic metric to evaluate the optimal checkpointing interval, and adopt the equidistant or equicost checkpointing strategies. A common assumption is that the execution time of the target program is known in advance.

[19] proposed a first-order approximation to the optimum checkpoint interval. The author assumes a system in which a failure is detected as soon as it occurs, the checkpointing interval is fixed, the checkpointing time is constant, and no failures occur during error recovery. In addition to these assumptions, the author adopted the equidistant strategy and assumed that the occurrence of failures are essentially random (a Poisson process), with the failure rate λ. Then the mean time T_f between failures is $T_f = 1/\lambda$, and the density function $P(x)$ for the time interval of length x between failures is given by $P(x) = \lambda e^{-\lambda x}$. This failure assumption has been used by most of the papers [5, 6, 7, 17, 20].

In [16], the authors relax the above assumptions in three ways: by considering general failure distributions, by allowing checkpointing intervals to depend on the reprocessing time and the failure distribution, and by allowing failures to occur during checkpointing and error recovery. They first discussed the equidistant checkpointing strategy and found that the system availability resulting from using the strategy depended only on the mean of the failure distribution. Then, the equicost strategy was introduced which is a failure-dependent and reprocessing-independent checkpointing strategy. For Weibull failure distributions, the authors showed that the equicost strategy achieved higher system availability than the equidistant strategy.

Instead of using the execution time as a metric, in [20], the authors presented an online algorithm for the placement of checkpoints. This algorithm keeps track of the size of the state of a program and a checkpoint is made when it is small.

Solutions have also been developed for parallel and distributed systems [13, 17] and mobile computing systems [5]. In [13, 17], the optimal checkpointing interval in synchronous checkpointing for multiple processes is considered based on a mean failure time. In [5], the authors derived an approximation to the optimal message number interval between checkpoints. In mobile computing environments, as part of the total application execution time, messages passing time is affected by link bandwidth, making it difficult to predict the execution time of a program. Therefore, to determine the checkpoint placement the authors utilized the received computational message number. It is assumed that the inter-failure time is exponentially distributed.

A common characteristic of all of these works is that, in the system model, the execution of the programs to be checkpointed is continuous and has a long execution period, and a uniform failure rate during the entire execution of the program is known in advance. In our study of MA systems, however, the execution of an MA is discrete in time because the MA

executes for a while at a host, and then stops execution to migrate to another host. This results in a quite different system model which required new solutions.

3. System model

As mentioned above, algorithms for checkpoint placement in conventional computer systems cannot be ported directly to MA systems because the system model of MA system is different from that of conventional systems. In an MA system, the MA carries out its assigned tasks on the hosts along its itinerary. The tasks are separated by the migration operations.

Within each migration operation, the MA terminates its execution on the previous host and prepares for the migration. The preparation includes releasing the allocated resources (stack, memory) and packing the code and data sections of the MA into an image. Then the image will be transmitted to the next host. When the next host receives the image, it will perform system defined checking (i.e., CRC checking) to guarantee that the image is not damaged during the transmission, and incarnate the image to a new MA if the image passes the checking. The new MA will continue the execution on this new host. Since the MA is executing on a new host, the execution environment is totally new and has no any relation with the previous host; the stack and memory for this agent is reestablished. Therefore, we can claim that the failure of the MA on the current host is independent with its failure on the previous host. This characteristic corresponds exactly to the Markov chain property. Consequently, the execution of an MA in an MA system can be modelled as a discrete-parameter Markov chain. In the following subsections, we first define this model, and then propose checkpoint placement algorithms for MA systems.

We consider an MA system model where a single MA executes and migrates along a predefined or self-initiated itinerary. With a predefined itinerary, the agent knows all the hosts that it will visit, while with a self-initiated itinerary, the agent only knows the first host it will visit and the following hosts are determined by the execution results on the previous host. The itinerary consists of N hosts, $Host_0$, $Host_1$,…, $Host_{N-1}$, and a home node, *Home*. *Home* launches the MA to $Host_0$ reliably, so we consider that the MA starts its execution on $Host_0$.

We assume that a mobile agent can take an independent checkpoint right after it lands on a host and before starts its execution. It cannot take a checkpoint during its execution on a host. After the agent finishes its execution on a host, it will migrate to the next host according to the itinerary. This process will continue until all the hosts have been visited. During the migration, only the code and data (computing results) of an MA on previous host will be transmitted to the next host. On the new host or during the recovery process from a checkpoint, the MA is incarnated and its execution environment is reconstructed. Based on these observations, we assume that all the failures of an MA are independent from each other. Accordingly, the execution of an MA is modelled as a discrete-parameter Markov Chain.

For simplicity, we assume that the stationary transition probabilities of this Markov Chain are fixed. Therefore, it is a homogeneous discrete-parameter Markov Chain. Let I be the state space and T be the parameter space, both are finite and discrete.

$$I = \{0, 1, 2, \ldots N+1\},\ N>1; \quad T = \{0, 1, 2, \ldots\}.$$

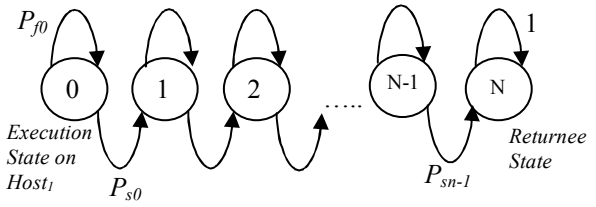

Figure 1 State Transition Graph

Figure 1 illustrates the state transition graph for an MA's execution. In state space I, state i ($0 \leqslant i \leqslant N-1$) denotes the *Execution State* of an MA on $Host_i$. Failure may happen during the execution and migration of the MA with probability P_{fi} on $Host_i$. P_{fi} is independent from each other. A failure is detected immediately and the agent recovers and starts the re-execution from the latest checkpoint with probability 1. Such a failure-recovery process is called a *failure-recovery round*, which may happen X_i times on $Host_i$. X_i is a random variable and its probability distribution is: $P(X_i)=(P_{fi})^{Xi}$.

The probability of successfully finishing the execution on the current host and landing on a new host is P_{si} ($P_{si} = 1 - P_{fi}$). P_{si} is independent from each other. The last state $N+1$ is the *Returnee State*, which means that the MA has returned home. In *Returnee State*, we assume that the MA is able to recover under the user's control with probability 1, no matter what failure occurs.

The purpose of a checkpoint placement algorithm is to determine the proper (or optimal) checkpointing interval so as to reduce (or minimize) the system cost, which consists of the cost of making checkpoints and the cost of recovery when failures happen. The

recovery cost includes the cost of incarnating the new MAs from the checkpoints and the re-execution cost. Table 1 shows the notions for the various types of cost in this model.

Table 1 Various Types of Cost

C_{sys}	Overall system cost
C_i	Recovery cost in one fail-recovery on $Host_i$
C_{cp}	Cost of making a checkpoint
C_I	Cost of incarnating an MA from a checkpoint
E_i	Cost of the execution on $Host_i$
C_{Xi}	Re-execution cost on $Host_i$

C_{cp} is the cost to store a checkpoint on disk while C_I is the cost to read a checkpoint from the disk. Although the direction of the data flow is different, the cost is similar: both of them can be evaluated by the cost of I/O operations and have no relation with specific applications. However, E_i is related with the specific application on $Host_i$ because different application has different execution cost. Therefore, we assume that C_{cp} and C_I are known in advance. E_i is provided by the MA platform of $Host_i$. P_{fi} is also maintained by the MA platform of $Host_i$. An MA does not know E_i and P_{fi} before it retrieves them from the MA platform of $Host_i$.

4. Checkpoint placement algorithms

The algorithms proposed in the next section are based on the equidistant and equicost checkpointing strategies. The principle of these checkpointing strategies is to seek a better balance between the expected recovery cost and the checkpointing cost. An optimal checkpointing interval can be achieved in conventional systems if the failure rate is the same during the entire execution duration of a program. Similarly, we can also get the optimal checkpointing interval for an MA if all the failure rates P_{fi} are the same. Otherwise we cannot get the optimal checkpointing interval. In our model, checkpointing cost C_{cp} is known in advance, so we just need to derive the expected recovery cost within a checkpointing interval to seek the balance between C_{cp} and recovery cost. In following, we firstly determine the optimal checkpointing interval for an MA under an ideal condition (all the P_{fi} are the same), and then using the result to derive heuristics for designing checkpoint placement algorithms in a realistic MA system, where P_{fi} are independent in nature.

4.1 Recovery cost in a checkpointing interval

With the reference to Figure 2, suppose an agent takes a checkpoint on $Host_i$ before it initiates its execution, and the agent takes its next checkpoint on $Host_{i+n+1}$. The checkpoint interval is defined as the number of hosts between $Host_i$ and $Host_{i+n}$ (including $Host_i$ and $Host_{i+n}$). The probability of the MA failure on $Host_i$ is P_{fi}, and the number of failure-recovery rounds is X_i with $P(X_i)=(P_{fi})^{Xi}$. The expected number of failure-recovery rounds (R_i) is given by Equation 1:

$$R_i = E(X_i) = \sum_{r=1}^{\infty} r * (P_{fi})^r = P_{fi}/(1-P_{fi})^2 \quad (0 \leqslant i \leqslant N-1) \quad (1)$$

The recovery cost C_i on $Host_i$ consists of the incarnation cost C_I and the re-execution cost C_{xi}, which refers to the cost of the execution starting from the checkpointing point to the failure point. Since the failure point is evenly distributed in the duration of an MA's execution on a host (as shown in Figure 2), the expected re-execution cost in one failure-recovery round is given by Equation 2.

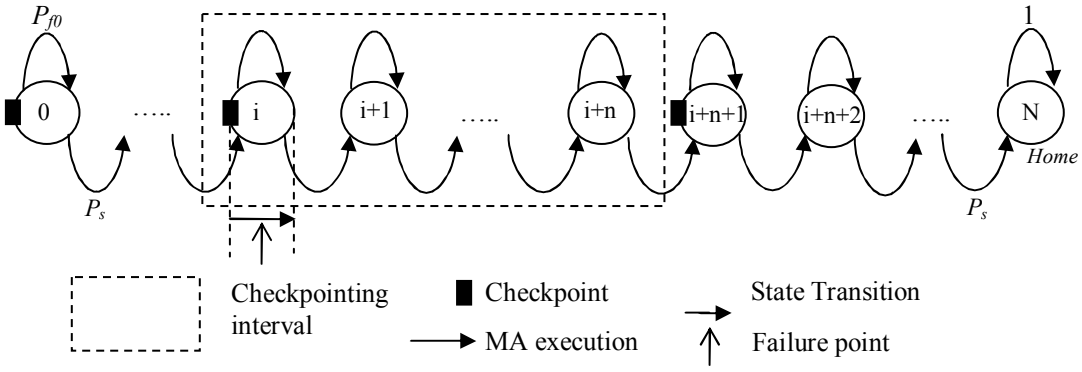

Figure 2 The Calculation of System Cost in a Checkpointing Interval

$$E(C_{Xi}) = \int_0^{E_i} x(dx / E_i) = (1/E_i)\int_0^{E_i} xdx = E_i/2 \quad (2)$$

Accordingly, the expected recovery cost on $Host_i$ is:

$$E(C_i) = R_i*(C_I + E_i/2) \quad (3)$$

For a failure-recovery round on $Host_{i+1}$, since the recovery should start from $Host_i$, the expected recovery cost on $Host_{i+1}$ is: $E(C_{i+1}) = R_{i+1}*(C_I+E_i+E_{i+1}/2)$ and the expected recovery cost on $Host_{i+n}$ is: $E(C_{i+n}) = R_{i+n}*(C_I+E_i+\ldots+E_{i+n}/2)$.

Since the failures are independent, the total expected recovery cost for the checkpoint interval illustrated in Figure 2 is given by Equation 4 below ($C_{i,i+n}$ denotes the total expected recovery cost from $Host_i$ to $Host_{i+n}$):

$$E(C_{i,i+n}) = E(\sum_{k=0}^{n} C_{i+k}) = \sum_{k=0}^{n} E(C_{i+k}) \quad (4)$$

Together with the cost for the checkpointing and execution cost within this interval, the overall system cost can be calculated by Equation 5:

$$C_{sys} = C_{cp} + E(C_{i,i+n}) + (n+1)E_i \quad (5)$$

4.2 The Optimal Checkpointing Interval under an Ideal Condition

If the cost of checkpointing and the failure rate are fixed during the execution of a program, the optimal placement strategy would be to place the checkpoints in fixed equidistant intervals [16, 20]. In our model, C_{cp} is fixed, but P_{fi} are different from each other (the same to E_i). To derive the heuristic rules for checkpoint placement algorithms in a realistic MA system where P_{fi} are independent in nature, we first consider an ideal condition: all the P_{fi} and E_i are the same.

Fixing P_{fi} makes the MA system to have a fixed failure rate. Having E_i with the same value on all the hosts makes the intervals equidistant if each interval contains the same number of hosts. Since we do not consider checkpointing in the middle of an MA's execution on a host, as shown in Figure 3, the granularity of the checkpointing interval is one host. To determine the optimal interval, we assume that an interval contains x hosts. Then the expected total system cost is calculated as follows:

$$C_{sys} = (H/x)[x*C_I + (RE+(x-1)RE)(x-1)/2 + xER/2 + C_{cp}] + H*E = (H/x)[x*C_I + x^2RE/2 + C_{cp}] + H*E = H*C_I + xHRE/2 + HC_{cp}/x + H*E \quad (x=1,2,3,\ldots N) \quad (6)$$

To get x that produces minimal value for Equation (6), we consider that x is continuous and then we can get x by derivative.

$$C_{sys}' = REH/2 - C_{cp}H/x^2 \quad (7)$$

$$C_{sys}'' = 2\,C_{cp}H/x^2 \quad (8)$$

Since $C_{sys}''>0$, C_{sys} has the minimal value. Let $C_{sys}'=0$, we can get the value of x to make C_{sys} minimal.

$$C_{sys}' = REH/2 - C_{cp}H/x^2 = 0$$

$$x = \sqrt{2C_{cp}/RE} \quad (10)$$

With the optimal x derived from Equation 10, the cost of the expected total re-execution cost in an interval (Figure 3) can be calculated using Equation 11 below.

$$\sum_{i=1}^{x} C_{Xi} = (RE+(x-1)RE)(x-1)/2 + xER/2 = C_{cp} \quad (11)$$

Equation 10 implies that the optimal checkpointing interval is only related with C_{cp}, R and E, and has nothing to do with C_I. Equation 11 tells us that within an optimal checkpointing interval, the expected total re-execution cost (variable part in Figure 3) equals exactly to C_{cp}. The implications can be used as heuristic rules in designing checkpoint placement algorithm based on the equicost strategy. However, notice that Equation 11 is gotten under the assumption that x is continuous, but x is actually discrete.

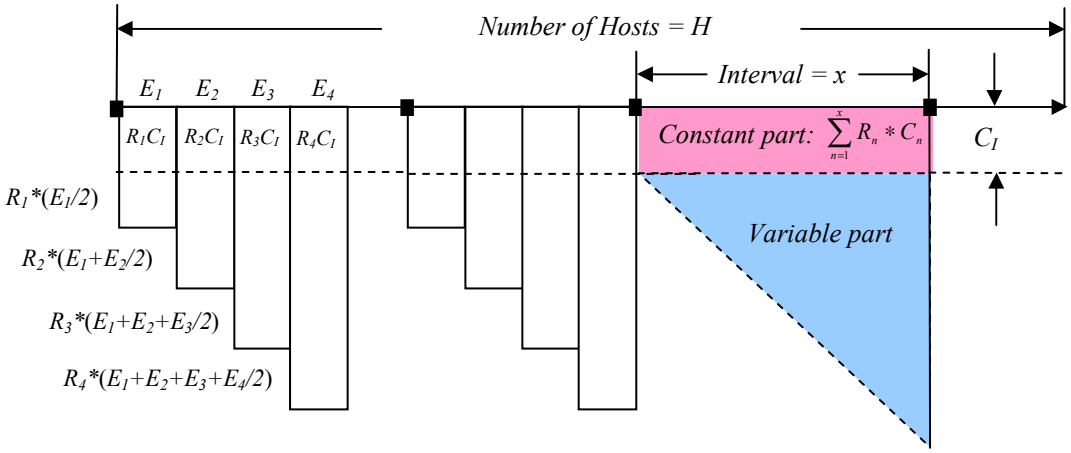

Figure 3. The cost in a Checkpointing Interval

Therefore, the optimal value should be the integer that is neighboring *x*. In Figure 4, the optimal interval should be *"3"* or *"4"*. We must determine the optimal interval by checking which value leads to a smaller result of Equation 6.

4.3 Checkpoint Placement Algorithms

In a real MA system, the failure rates cannot be the same on all the hosts and links, so we need to design algorithms working in general conditions. From the analysis in Section 4.2, we propose three checkpoint placement algorithms.

Algorithm 1: *Equidistant with failure rate estimation:* before an MA starts its travelling, it estimates a uniform failure rate for the MA system and decides the checkpointing interval by using Equation 10 in Section 4.2. The estimation can be made by using all the P_{fi} collected from the MA platforms on each host.

Algorithm 1:

```
itinerary= MA.GetItinerary(); // predefined Itinerary is needed;
Pf[ ] = MA.GetFailureProbability(itinerary); //Collect Pfi;
AvrPf = MA.AveragePf(Pf[ ]);    // Eastimat average failure rate;
R= AvrPf /Power(1- AvrPf);      //Get R according Equation 1;

OptInterval_float = Sqrt(Ccp/R*E); //get interval (Equation 10);
OptInterval_Ceiling=Ceiling(OptInterval_float);
OptInterval_Floor=Floor(OptInterval_float);
//(Calculate the system cost according to Equation 6);
OptC=OptInterval_Ceiling*H*R*E/2+H*Ccp/OptInterval_Ceiling
OptF=OptInterval_Floor*H*R*E/2+H*Ccp/OptInterval_Floor;
//the integer leads to smaller cost becomes the checkpoint interval;
if (OptC< OptF)
  OptInterval= OptInterval_Ceiling;
else
  OptInterval= OptInterval_Floor;
MA.Context.CpInterval= OptInterval;
 MA.Migration(itinerary);    // every "OptInterval" hosts.
```

Algorithm 2: *Equidistant by enumeration:* before an MA start its travelling, it collects all the P_{fi} maintained by the MA platforms on the hosts, and enumerate the results of Equation 6 with the value of *x* from *1* to *H* to get the *x* that leads to the smallest result.

Algorithm 3: *Equicost:* MA calculates the variable part (C_{var}) of the re-execution cost before its execution on each host. A checkpoint is made when C_{var} is equal to C_{cp} or greater than C_{cp} (according to the heuristic rules described in Section 4.2).

The pseudo-code of these algorithms is shown in Appendix.In Algorithm 1, to collect P_{fi} from all the hosts along the itinerary, a predefined itinerary is needed. Since we require only an estimated average failure rate, the interval obtained here is only an approximate optimal interval. In Algorithm 2, a

Algorithm 2:

```
itinerary= MA.GetItinerary(); // predefined Itinerary is needed;
Pf[ ] = MA.GetFailureProbability(itinerary); //Collect Pfi;

for i=1 to itinerary.NumberOfHosts //Calculate R (Equation 1)
   R[i]=Pf[i]/Power(1- Pf[i]);
   RC[i]=E* R[i];  //RC[i]: the re-exe cost on single host i;
end

for interval=1 to itinerary.NumberOfHosts //Enumerate cost;
   TC=Ccp; // Initiate the total cost;
   accumulateC=0; // Temp var: record re-exe cost;
   currentC=0;  //Temp var: count current cost;
   index=1; //Temp var: record how many host has passed;

   for host=1 to itinerary.NumberOfHosts
      for i=1 to index  //count the re-exe cost if fail;
        accumulateC=accumulateC+RC[host-i+1];
      end
      currentC = currentC + accumulateC;
      if (index==interval) //Pass_hosts= interval, cp is made;
         TC=TC+Ccp+currentC; //count the system cost;
         accumulateC=0;  //Restore the temp variable;
         currentC=0;   //Restore the temp variable;
         index=1;    //Restore the temp variable;
      else
         index=index+1; //Still not reach the interval boarder
      end
   end
   if (index<interval)   //reach the tail of the itinerary;
     TC =TC+currentC;
   end
   allresults[interval]=TC;
end
OptInterval = MiniIndex(allresults[]);
MA.Context.CpInterval= OptInterval; //MA will carry the interval
MA.Migration(itinerary);
```

Algorithm 3:

```
//MA initiation at home node;
MA.context={Ccp, E, FirstHost=TRUE, TC=0,
accumulateC=0, currentC=0, index=1};
MA.Launch(firsthost); // launched to the first host

//Landing Procedure;
if (MA.FirstHost== TRUE)
  TC=Ccp; //A checkpoint is made at the first host in an interval;
  MA.FirstHost== FALSE;
end
Pf=MAP.getPf();  //Pf is maintained by the MA platform on a host;
R=Pf/Power(1- Pf);  //Calculate R according Equation 1
RC[index]=E*R;  //RC[i]: re-execution cost on a single host i;

for i=1 to index  // re-exe cost if failure happen on this host;
  accumulateC=accumulateC+RC[i];
end
currentC = currentC + accumulateC; //re-exe cost for past hosts;
if (currentC>=Ccp) // re-exe cost >=Ccp, a checkpoint is made;
   TC=TC+Ccp+currentC; //record the system cost
  accumulateC=0; //Restore the temporary variable;
  currentC=0; //Restore the temporary variable;
  index=1; //Restore the temporary variable;
else
  index=index+1;  //Still not reach the interval boarder
  if (ReturnedHome=TRUE)  //reach the tail of the itinerary;
    TC=TC+currentC;
end
```

predefined itinerary also is necessary for the P_{fi} collection. All the possible intervals will be tried to get the best equidistant interval. Obviously, Algorithm 1 cannot give a better equidistant interval than Algorithm 2, but the time complexity of Algorithm 1 ($O(1)$) is much lower then that of Algorithm 2 ($O(H^2)$). Compare with Algorithms 1 and 2, the big advantage of Algorithm 3 is that it does not require a pre-defined itinerary. The decision on checkpointing is made during the execution of an MA. As mentioned in Section 1, a most prevailing characteristic of MA is the autonomy, which allows an MA to determine its itinerary dynamically. Algorithm 3 has no constraint for MA to maintain this characteristic.

5. Performance Evaluation

We need to evaluate the performance of the three proposed algorithms to find out which algorithm is the best in the sense that achieves the lowest system cost. Since we do not assume any distribution for the occurrences of failures on the hosts, there is no suitable way to make an analytic analysis. Therefore, we have carried out simulations to simulate the three algorithms using the Markov model shown in Figure 2.

In our simulations, we consider an MA system with 100 hosts. On host i, we consider two ranges of failure probabilities: $P_{fi}<0.01$ and $P_{fi}<0.001$. We adopt a uniform unit for the system cost, which can be the execution time or some other metrics. We assume $C_{cp} \geqslant 1$, E $\geqslant 1$ and $C_l = C_{cp}$. The execution cost E can be less than C_{cp} if MA's tasks on hosts are short and have no I/O operations; otherwise E is larger than C_{cp}. Since the cost of the constant part (Figure 3) is the same for all the three algorithms, we only compare their differences on the variable part C_{var}. The results shown in the following Figure 4 are obtained with C_{cp} set to 2 and the execution cost E ranging from 1 to 20. For each cost metric, the same simulation is performed 100 times to get an average value for the checkpointing interval and the variable part C_{var}.

Figures 4 illustrate the average checkpointing interval and corresponding variable part cost for the three algorithms under different failure probability ranges. A general tendency observed is that, with the failure probability decreasing, the checkpointing

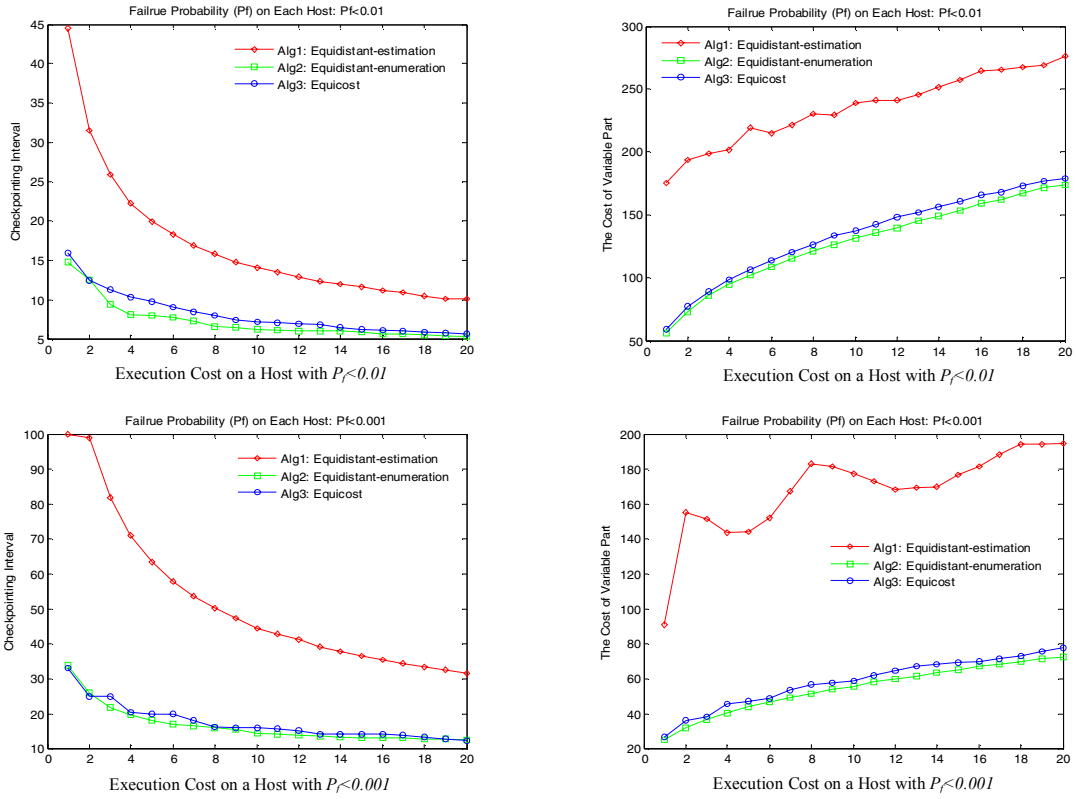

Figure 4 Checkpointing Interval and Corresponding Cost for Variable Part

interval will become bigger. In terms of the system cost, Algorithm 2 (equidistant by enumeration) gives the best performance, while Algorithm 1 (equidistant with failure rate estimation) is always the worst. Algorithm 3 (equicost) leads to similar system cost with Algorithm 2, but it is much more flexible and suitable for an MA system as we discussed at the end of the last section.

6. Conclusions and Future Works

MA system has a different system model from conventional computer systems in solving the checkpoint placement problem. In this paper, we have proposed to model the problem for MA systems using the homogeneous discrete-parameter Markov chain. Based on this model and two widely adopted checkpointing strategies, we designed three checkpoint placement algorithms for MA system. Through simulations we found out that the algorithm based on equicost checkpointing strategy achieved the best trade-off between checkpointing cost, system performance, and flexibility.

In this paper, we only considered the singe MA's independent checkpointing. However, coordinated checkpointing is needed for a group of mobile agents. How to determine the optimal checkpoint placement in an MA group is our future work.

Acknowledgement

This work is supported in part by the University Grant Council of Hong Kong under the CERG Grants PolyU 5075/02E and PolyU 5183/04E.

References

[1] J. Cao, G.H. Chan, W. Jia, and T. Dillon, "Checkpointing and Rollback of Wide-Area Distributed Applications Using Mobile Agents", Proc. IPDPS2001 - IEEE 2001 International Parallel and Distributed Processing, April 2001, San Francisco, USA.

[2] D. Chess, C. harrison, and A. Kershenbaum, "Mobile Agents: Are They a Good Idea?", IBM Research Report, RC 19887 (#88465) 3/16/95.

[3] D.B. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents", Communication of the ACM, Vol. 42, No. 3, March 1999. pp. 88-89.

[4] E. N. Elnozahy, D. B. Johnson, Y. M. Wang. "A Survey of Rollback-Recovery Protocols in Message Passing Systems", ACM Computer Surveys, Volume 34, Number 3, September 2002 pp. 375-408

[5] Xinyu Chen and Michael R. Lyu "Performance and Effectiveness Analysis of Checkpointing in Mobile Environments" Proceedings of the 22nd International Symposium on Reliable Distributed Systems (SRDS'03) Florence, Italy, 131-140

[6] E. Gelenbe and D. Derochette, "Performance of Rollback Recovery Systems under Intermittent Failures," CO". ACM 21(6) pp. 493-499 (June 1978).

[7] E. Gelenbe, "On the Optimum Checkpoint Interval," Journal of the ACM 26(2) pp. 59-270 (Apr. 1979).

[8] K. Mani Chandy, James C. Browne, Charles W. Dissly, and Werner R. Uhrig, "Analytic Models for Rollback and Recovery Strategies in Data Base Systems," IEEE ransactions on Software Engineering SE-l(1) pp. 100-1 10 (March 1975).

[9] A. Mohindra, A. Purakayastha, and P. Thati, "Exploiting nondeterminism for reliability of mobile agent systems," in Proc. Int. Conf. Dependable Systems Networks, Los Alamitos, CA, 2000, pp. 144–153.

[10] Victor F. Nicola and Johannes M. Van Spanje, "Comparative Analysis of Different Models of Checkpointing and Recovery," IEEE Trans. Software Engineering 16(8) pp. 807-821 (Aug. 1990).

[11] Oliner, A.J.; Sahoo, R.K.; Moreira, J.E.; Gupta, M.; "Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems". Proceedings. 19th IEEE International Symposium on Parallel and Distributed Processing, 04-08 April 2005 Page(s):299b - 299b

[12] Taesoon Park; Ilsoo Byun; Hyunjoo Kim; Yeom, H.Y.; "The performance of checkpointing and replication schemes for fault tolerant mobile agent systems" 2002. Proceedings. 21st IEEE Symposium on Reliable Distributed Systems, 13-16 Oct. 2002

[13] J. S. Plank and W. R. Elwasif. "Experimental assessment of workstation failures and their impact on checkpointing systems". In 28th International Symposium on Fault-Tolerant Computing, pages 48–57, Munich, June 1998.

[14] Gyung-Leen Park; Hee Yong Youn; Hyun-Seung Choo; "Optimal checkpoint interval analysis using stochastic Petri net". Proceedings. 2001 Pacific Rim International Symposium on Dependable Computing. 17-19 Dec. 2001 Page(s):57 – 60

[15] Kang G. Shin, Tein-Hsiang Lin, and Yann-Hang Lee, "Optimal Checkpointing of Real-Time Tasks," IEEE Transactions on Computers C-36(11) pp. 1328-1341 (NOV. 1987).

[16] Asser N. Tantawi and Manfred Ruschitzka, "Performance Analysis of Checkpointing Strategies." ACM Transactions on Computer System 2(2) pp. 123-144 (May 1984).

[17] K. F. Wong and M. Franklin. "Checkpointing in distributed systems". Journal of Parallel & Distributed Systems, 35(1):67–75, May 1996.

[18] Chenggang Wu; Shaohui Liu; Bo Wang; Zhongzhi Shi; Hua Gu; "Configurable mobile agent and its fault-tolerance mechanism" Proceedings of International Conference on Computer Networks and Mobile Computing, 16-19 Oct. 2001 Pages: 380-389

[19] John W. Young, "A First Order Approximation to the Optimum Checkpoint Interval," Communications of the ACM 17(9) pp. 530-531 (Sept. 1974).

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