Integrated Replication-Checkpoint Fault Tolerance Approach of mobile agents "IRCFT"

Suzanne Youcef Sweiti and Amal Moh'd Al. Dweik
College of Information Technology and Computer Engineering
Palestine Polytechnic University
Hebron, Palestine
suzans@ppu.edu amal@ppu.edu

Abstract: *Mobile agents offer flexibility which is evident in distributed computing environments. However, agent systems are subject to failures that result from bad communication, breakdown of agent server, security attacks, lack of system resources, congestion in network, and situations of deadlock.*
If any of such things happen, mobile agents suffer loss or damage totally or partially while execution is being carried out. Reliability must be addressed by the mobile agent technology paradigm. This paper introduces a novel fault tolerance approach "IRCFT" to detect agent failures as well as to recover services in mobile agent systems. Our approach makes use of checkpointing and replication where different agents cooperate to detect agent failures. We described the design of our approach, and different failure scenarios and their corresponding recovery procedures are discussed.

Keywords: *Mobile agents, fault tolerance, reliability, checkpointing, replication.*

1. Introduction

In the both the academic and industrial trends, mobile agents are very important in the recent trends of distributed computing. They have their own properties which make them flexible in the deployment. As a result, the design, application and maintenance of the distributed systems become a simple task [1,2].
Like any other software systems, mobile agents are not isolated from operating in unusual situations. They are somewhat subject to fault as they consist of autonomous components in distributed dynamic environments [3]. Mobile agents might come across usual errors which emerge especially during migration request failure, security penetration or communication exceptions [4]. The issue of reliability is extremely important in order to challenge such failures. The goal is to allow the system to work flexibly in spite of the faults which continue to exist in the system after development [2].
The most important requirements among the approaches to handle the fault tolerance in mobile agent system (MAS) are: non-blocking and exactly-once [5].
The non-blocking property guarantees that the agent continues executing to achieve its goal even in case of an infrastructure component.
The use of replication may cause the violating of the dominant property of the mobile agent: the exactly once execution of the mobile agent [5]. The objective of the protocols for dealing with the exactly-once property is to ensure that the agent has to carry out the intended action one time only in a host.
The remainder of this paper is organized as follows: the next section introduces the previous work. Section 3 defines our model while section 4 introduces the different failure scenarios and their corresponding recovery procedures. Implementation is introduced in section 5. Finally, the conclusion is introduced in section 6 followed by the references.

2. Related Work

In this section, some existing fault tolerance approaches are briefly discussed.

In [6], the authors have proposed an approach for providing efficient fault tolerance in mobile agent systems to overcome certain failures. Here, the parallel checkpointing approach is used which considers the antecedence graphs. The used graphs are directed acyclic graphs that record the dependency information. The identifying time is minimized and time latency is decreased for global checkpointing procedure, as during the beginning the relevant mobile agent is being informed by the initiator simultaneously. In this technique [6], the identifying, execution and recovery time is improved and the message overhead can be reduced. On the other hand, there is an overhead of taking message log and antecedence graph of each message.

The authors of [7] have proposed an approach to sort out the agent crash problem. Here the clone of original agent is created. This clone is used in an itinerary to follow the actual agent. So, if any failure occurs in the mobile agent system, the recovery is possible by the clone. The main aim is to limit the rollback by adding checkpoints. The performance can be improved by total trip time, checkpoint time and successful migration time. However, it may sometimes violate the exactly once property of mobile agents.

In [8], authors introduce that the server and agent failures are detected and recovered by the cooperation of agents with each other. In order to detect and recover the failed agent in 2-Dimensional Mesh Network , another types of agent are used, namely the witness agent, to monitor whether the actual agent is alive or dead. It prevents a partial or complete loss of mobile agent. In this approach the use of 2-D mesh network, dependencies among witness agent get reduced as compare to linear network and the drawback is that the existing procedure consumes a lot of resources along the itinerary of the actual agent as the itinerary becomes longer, more witness agents and probes are necessary, so system complexity increases.

In [9], the idea of agent tracking technique is described. It provides an efficient system so that excellent management and maintenance can be performed in the networks of the systems of mobile agents. This technique is claimed to be beneficial in Maintenance and Efficiency which are attained through robust tracking techniques. The drawback of this approach is that it doesn't comprise replication and techniques for timely fault detection. These properties promote the robustness and flexibility in the system.

In [10], the fault tolerant mechanism has been laid with relation to the applications that deal with transactions. The protocol proposed is based on the behavior of mobile agent, Watch Agent as well as Transaction Manager. When commit at destination protocol is applied, the property of exactly-once execution is not violated.

The designing of adaptive mobile agents in [11] aims at accepting additional roles when working inside a special environment that is called context-aware environment. The merit of the technique is that the mobile agents are already inside the system. So, there is no need for any kind of external communication. This means that the time to make and send a new mobile agent is saved and time for response becomes shorter.

In [12], an integrated mechanism has been proposed using SMAPS (Secure Mobile Agent Platform System) which aims to prevent agent blocking in certain cases where agent is identified by malicious host. When executing transactions, the partial result retrieval is taken into account. This is then used to follow the location of the mobile agents during the process at any time. The technique in [13] is helpful in improving the reliability and performance through parameters like communication overhead, size of the message and fault tolerance time. This technique's shortcoming is that is suitable merely for time sensitive applications.

In [13], the agent put its computation results on the home server after completing its task on first three servers in its itinerary. The approach makes use of check pointing, partial results and the address of last host visited is saved prior before the agent visits the next host in the itinerary. If agent stops its execution due to any fault on the server and unable to move in its itinerary, agent sends a message to home server, which then sends the replicated copy of the original agent to the immediate checkpoint before the faulty server. The authors measure the performance of the approach. The analysis of this technique show a good result by improving the round trip time of the agent, since after occurrence of fault, the replicated agent need not roll back to the first server as it starts moving from the checkpoint immediately before the faulty server. Whenever an agent does not reaches the desired server due to network congestion the host assumes it to be failed and it sends a replicated copy of it means while the original agent also reaches the destination which could lead to violation of exactly once property.

In [14], the authors raised another Fault-Tolerant model based on Witness. This model employs three types of agents. They are actual agent which performs programs for its owner, witness agent which monitors the actual agent and the witness agent after itself, probe which is sent for recovery the actual agent or the witness agent on the side of the witness agent. The older witness agents are responsible for monitoring the witness agent that is just one server closer to the actual agent in its itinerary. Communication mechanism is message passing between these agents. Discovery of failure is done by witness agent and recovery is message log based recovery and also checkpointing. The improvement in agent survivability is achieved by spending more time and space resource's, more witness agents and probes are necessary, so system complexity increases.

3. Integrated Replication-Checkpoint Fault Tolerance Approach (IRCFT)

The IRCFT approach is a development of that implemented in [13] and [14]. It provides the availability of recovery mechanism in the suggested approach, the execution proceeds after the crash of the agent or crash of the host smoothly and correctly. The basic idea used in the work is to tolerate faults using

the concept of checkpoint and replication. For our work we will make the following enhancements:

- The key difference between the protocol suggested in [14] and our protocol is that: the former depends on a reliable network, while we allow the network to be unreliable. Therefore, we can handle the failures in transmission of messages as well as the loss of the agent in the network.
- Compared with work [14] our approach depends on using fewer monitors to monitor the worker agent during work. Connection is performed between four monitors maximum in each stage to detect the error when it occurs in the worker agent. So the previous monitors are killed after doing the checkpoint in the home server.
- Our approach provides the availability of recovery mechanism in the suggested approaches, the execution proceeds after the crash of the agent or crash of the server smoothly and correctly.

3.1. Assumptions

Some specific assumptions in the system may be summarized in the following points:

1. Agents in the system can be generated from every node on network. Thus, each node on network provides the mobile agents an execution environment to accomplish its tasks.
2. There is no implementation of platform crash or platform failure. However, the crash of the server, is treated somehow by treating it as crash of the agents they are hosting.
3. The home server is always available.
4. There is no case where all servers are down at the same time.
5. No hardware failure in log message, such that cannot be recorded in the permanent storage.

3.2. IRCFT Description

In our approach, we distinguish five types of agents, one type is performing the required computation for the user. We name it the worker agent (WA). WA play an active role in this protocol. Another type is to detect the status of the actual agent. We call it the monitor agent (MA). Manager Agent that send the address of the next server to the WA, and the fourth type is the replica to recover the WA. MA and WA communicate by using a peer-to-peer messages passing mechanism. MA sets a timer with a certain time-out value for each server S_i. It creates Repair agent (RA) in some case of monitor failure. Replica is used to protect the WA. We also need to log the actions performed by the WA. We use checkpoint data to recover the lost agent, and save partial results before the agent visits the next server in the itinerary.

At first, the Manager agent create WA and MA and they migrate to the next server Figure 1 illustrates the workflow of the protocol.

Suppose that, currently there are *n* network nodes on the execution itinerary, and the WA does not reach the fourth server yet, WA and MA are at server S_{i+1}.

1- After WA has arrived at S_{i+1}, it immediately registers a log_{arrive} on the permanent storage in S_{i+1}.

2- Afterwards, WA_{i+1} informs MA_{i+1} that it has arrived at S_{i+1} safely by sending a msg_{arrive} message to MA_{i+1}.

3- MA_{i+1} informs MA_i that WA_{i+1} has arrived at S_{i+1} safely by sending a msg_{arrive} message to MA_i.

4- Next, WA_{i+1} accomplishes the task appointed by the owner on S_{i+1}.

5- WA_{i+1} takes a checkpoint to a secondary storage in the resident server in timeout period (t_{ck}).

6- WA_{i+1} sends message checkpoint_msg to MA_{i+1}. If the checkpointing action fails, the worker agent will abort the whole transaction.

7- WA_{i+1} sends its current address to the Manager agent and waits for a response for address of the next server. If not response resends the message.

8- Manager agent sends address of next server if the current address is not in the list of the precedent address visited.

9- WA_{i+1} registers a log_{leave} in S_{i+1}.

10- Next, WA_{i+1} sends to MA_{i+1} a message, msg_{leave}, in order to inform MA_{i+1} that WA_{i+1} is ready to leave S_{i+1}.

11- MA_{i+1} sends message msg_{leave} to MA_i.

12- MA_{i+1} creates a new monitor in S_{i+1}.

13- WA_{i+1} and the new monitor leave S_{i+1}, and migrates to S_{i+2}.

14- MA_{i+1} creates a new replica agent RcA_{i+1}.

15- RcA_{i+1} go to the checkpoint and change it status.

16- RcA_{i+1} sends an update message to the previous replicas and waits for reception of acknowledge (ack) message from the other replicas.

After completing its execution on the first three servers of the itinerary the worker move to S_{i+2} and repeats the following stapes 1-4 and 7-11 above motioned.

When WA_{i+2} sends the log_{leave} to MA_{i+2}, then:

17- WA_{i+2} move back to the home server and checkpoint the data and save the values computed from the previous three servers.

18- Once the WA completes checkpoint in the fourth server the replicas and monitors are killed since it is no longer needed.

19- After saving the value and adding checkpoints the Manager Agent creates a new monitor. WA_{i+2} and the new monitor move to the next server in the itinerary that is S_{i+3} and repeat the same process for every four servers in the itinerary.

- A chain of monitors is built in the itinerary $MA_{i-1} \rightarrow MA_i \rightarrow MA_{i+1}$. The monitor MA_i sends a message to monitor MA_{i-1} periodicaly. Similarly, the monitor MA_{i+1} sends a message to the last minitor MA_i.

Finally, the WA_n completes its itinerary and accomplishes its objectives, it returns back to the home server.

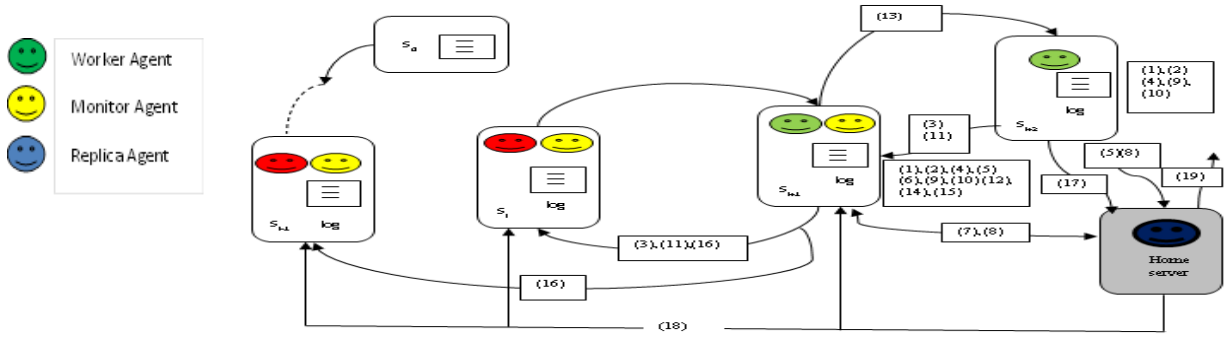

Figure 1. Workflow of IRCFT protocol.

3.3. Preserving exactly-once property

Exactly-once property must be preserved in the proposed approach. Because it is impossible for more than one worker to gain the accept from the Manager agent for the same stage, and it never receive the next server. In this case the duplicate agent will be terminate.

3.4. Management of monitors

In [14], monitor will be recreated when it is failed. This introduces a high overhead in the computation power when there is a host with a high crash rate. In this approach, the monitors will not be recreated but reconnection is used to maintain the chains of monitors. Besides the lower consumption of power, it can eliminate the "bad hosts" from the chain.

4. Failure Scenerios

In following subsections, we will cover different kinds of failures including the loss of the WA. We proposed several scenarios as follows.

4.1. In the first three servers

Figure 2 illustrates the whole execution life cycle in the first three servers and the arrow means where the failure occurs. The current host is S_{i+1}.

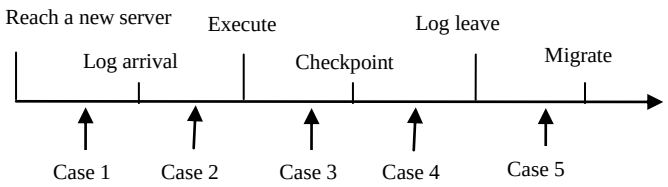

Figure 2. Different scenarios of failure in the first three servers.

4.1.1. Safe case

The MA_i successfully receive msg_{arrive} and msg_{leave} from MA_{i+1} and the worker agent successfully completes execution on S_{i+1} and moves safely to the next server S_{i+2}. The algorithm of the safe case is shown in Figure 3.

```
If (msg is valid)
    If (MAi receive msgarrive)
        Write (receive arrive)
        Else // If (receive msgleave)
            Write (MAi receive leave)
    EndIf
EndIf
```

Figure 3. Algorithm of the safe case.

4.1.2.The monitor MA_{i+1} fails to receive msg_{arrive}

Case 1 and 2 : The reason can be WA_{i+1} is dead when it is ready to leave Si, WA_{i+1} is dead when it has just arrives at S_{i+1}, or is dead when it has just arrived at S_{i+1} with logging.

In this case MA_{i+1} waits for the message msg_arrive for timeout period. If the timeout is reached it sends to MA_i failure message failure_msg. Then MA_i sends failure_msg to RcA_i and travel to S_{i+1} to recover the failure. Figure 4. show the algorithm of case 1 and 2.

```
If (! receive msgarrive) //MAi+1 fail to receive msgarrive
    MAi+1 send failure_msg to MAi
        If (MAi receive failure_msg)
            MAi send failure_msg to RcAi
            Migrate RcAi to the faulty server
        EndIf
EndIf
```

Figure 4. Algorithm of case 1 and 2.

4.1.3. The monitor MA_{i+1} fails to receive checkpoint message

Case 3and 4 : The reason can be WA_{i+1} is dead when it has just sends msg_arrive to MA_{i+1}, finishes it execution or do the checkpoint.

In this case MA_{i+1} waits for the checkpoint_msg for timeout period. If the timeout period is reached, it verify the checkpoint, if the checkpoint is Null it sends

to MA_i a failure message failure_msg. Then MA_i send failure else it sends to MA_i an error message error_msg. Then MA_{i+1} creates a new replica agent RcA_{i+1}. RcA_{i+1} changes its status according to the checkpoint to recover the failure. Figure 5 show the algorithm of case 3 and 4.

```
If (! receive checkpoint_msg)
   If (checkpoint == Null)
      MA_{i+1} send failure_msg to MA_i
         If (MA_i receive failure_msg)
            MA_i send failure_msg to RcA_i
            Migrate RcA_i to the faulty server
         EndIf
   Else
      MA_{i+1} send error_msg to MA_i
         If (MA_i receive error_msg)
            MA_{i+1} create RcA_{i+1}
            RcA_{i+1} change the status according to the
            checkpoint
         EndIf
EndIf
```

Figure 5. Algorithm of case 3 and 4.

4.1.4. The monitor MA_{i+1} fails to receive msg_{leave}

Case 5 : The reason can be WA_{i+1} is dead when it has just sends checkpoint_msg to MA_{i+1} or log_{leave}.

In this case MA_{i+1} waits for the msg_{leave} for timeout period. If the timeout is reached it sends to MA_i error message error_msg. Then MA_{i+1} creates a new replica agent RcA_{i+1}. RcA_{i+1} change its status according to the checkpoint to recover the failure. Figure 6 show the algorithm of case 5.

```
If (! receive msg_leave)
     MA_{i+1} send error_msg to MA_i
         If (MA_i receive error_msg)
            MA_{i+1} create RcA_{i+1}
            RcA_{i+1} change the status according to the
            checkpoint
         EndIf
EndIf
```

Figure 6. Algorithm of case 5.

4.1.5. The monitor MA_i fails to receive msg_{arrive} or msg_{leave}

The reasons can be the message msg_{arrive} or msg_{leave} is lost due to an unreliable network or arrives after the timeout period of MA_i or WA_{i+1} and MA_{i+1} are died.
For the first reason the WA_{i+1} does not die in S_{i+1}. In this case MA_i waits for the message for timeout period. If the timeout is reached MA_i sends Message-failure_msg to RcA_i and travel to S_{i+1} to search for log_{arrive} or log_{leave} in S_{i+1}. If found, then RcA_{i+1} retransmits msg_{arrive} or msg_{leave} to MA_i, then kill itself.
failure_msg to RcA_i and travel to S_{i+1} to recover the When WA_i fail to receive msg_{arrive}, it sends Arrive-Message-failure_msg to the RcA_i and travel to S_{i+1} to search for log_{arrive}. Upon arriving at S_{i+1}, it searches the log file in S_{i+1} for the entry log_{arrive}. If the log entry is not found, RcA_{i+1} sends a message to MA_{i+1} to verify if it is died or no. if RcA_{i+1} fails to receive the response, it creates a new monitor and recover the WA_{i+1}.
When WA_i fail to receive msg_{leave}, it sends leave-Message-failure_msg to the RcA_i and travel to S_{i+1} to search for log_{leave}. Upon arriving at S_{i+1}, it searches the log file in S_{i+1} for the entry log_{leave}. If the log entry is not found, RcA_{i+1} sends a message to MA_{i+1} and waits for a response. If it fails to receive the response, it creates a new monitor and searches the checkpoint. If checkpoint is found, it sends Worker-failure-AC_message to MA_i and changes the status according the checkpoint and sends its current address with error to the Manager agent and waits for a response for address of the next server and continue normally the remaining task. Figure 7 show the algorithm.

```
If (! receive msg_arrive) // or If (!receive msg_leave)
    If (! receive failure_msg)
        MA_i sends Message-failure_msg to RcA_i
        Migrate RcA_i to the S_{i+1}
            If (found(log_arrive))// or If (found log_leave)
                Retransmit msg_arrive to MA_i // or msg_leave
            EndIf
    EndIf
EndIf
```

Figure 7. Algorithm of message lost

4.1.6. WA_{i+1} and RcA_i fail

In this case MA_{i+1} sends failure_msg to MA_i. MA_i detects that RcA_i dies then, it sends failure_msg to MA_{i-1} in S_{i-1}. Then MA_{i-1} sends failure_msg to RcA_{i-1} and migrates to S_{i+1} to recover the failure.

4.1.7. Monitor fails

If the monitor MA_i is down, the monitor MA_{i-1} will exclude it from the chain by linking with monitor MA_{i+1}.
If MA_{i+1} receives Reconnect message from a monitor, it will reconnect to it.

4.2. In the fourth server

When the WA migrates to the fourth server and terminates the execution, the RcAs and the MAs in the previous servers will be killed, then if the WA stops its execution due to any fault when it is in the next server, in this situation immediately a fault message is sent from the MA to the home server and we have no other option than to send the replicated copy of the agent. The data retrieved from the previous serves is already saved to the home server with the checkpoints after

every four servers. So the replicated agent needs not to roll back to the first server in the itinerary.
The replicated agent is intelligent enough that it already knows the location of the fault and the immediate checkpoint before the fault. So it starts its execution from the immediate checkpoint.

5. Implementation

We are going to simulate the developed approach Integrated Replication-checkpoint Fault Tolerance Approach using AGLETS-2.0.2 and going to evaluate the agent survivability for failure recovery using the round trip time (time required by mobile agent to complete its itinerary) parameter. The works in [13] and [14] will be compared to our developed approach.

6. Conclusion

The IRCFT approach supposed to improve and enhance the survivability of the agent and it will diminish the time needed for detecting faults and repairing failure. Then the transmission of mobile agent to the next server will be more reliable. Furthermore, it will decrease trip time when errors occur.

The agent replicas and checkpoint are not executing while the original executing agent is active. Therefore, only one execution of the agent will be guaranteed at the same time. This property ensures the exactly once execution which is the most important feature for the agent execution. The non-blocking feature is also guaranteed even in the case of multiple failures by allowing the last checkpoint or the replica of the crashed agent to replace it in order to continue execution even in the case of agent failures. Hence, both checkpoint and replication is introduced in the suggested approaches to solve the blocking problem of the mobile agent execution where the replication and checkpoint masks failures and ensures progress of the mobile agent execution.

References

[1] W. Qu and H. Shen, "Analysis of Mobile Agents, Fault-Tolerant Behavior," IEEE/WIC/ACM, Proceedings of International Conference on Intelligent Agent Technology, 2004, pp 377 – 380, ISBN: 0-7695-2101-0.

[2] L. L. Pullum, "Software Fault Tolerance Techniques and Implementation," Artech House, 2001, ISBN 1-58053-137-7.

[3] W. Dake and C.P. Leguizamo and K. Mori, "Mobile Agent Fault Tolerance in Autonomous Decentralized Database Systems," IEEE, Proceedings of the Autonomous Decentralized System on The 2nd International Workshop, 2002, ISBN:0-7803-7624-2.

[4] G. Serugendo and A. Romanovsky, "Designing Fault-Tolerant Mobile System," Springer, International Workshop on Scientific Engineering for Distributed Java Applications, 2003, pp 185-201, ISBN: 978-3-540-00679-4.

[5] F. Bellifemine, G. Caire and D. Greenwood, "Developing Multi-Agent Systems with JADE," John Wiley & Sons Ltd, England. British Library Cataloguing In Publication Data, 2007.

[6] S. Rajwinder and D. Mayank, "Antecedence Graph Approach to Checkpointing for Fault Tolerance in Mobile Agent Systems", In Proc. of IEEE Transactions On Computers, Volume 62, issue 2, Publication IEEE Conference, February 2013.

[7] H. Rahul and K. Ramandeep, "Novel Dynamic Shadow Approach for Fault Tolerance in Mobile Agent Systems," In Proc. of 6th International Conference on Signal Processing Communication Systems, Publication IEEE Conference, 2012.

[8] A. Rostami, H. Rashidi, M. S. zahraie, "Fault Tolerance Mobile Agent System Using Witness Agent in 2- Dimensional Mesh Network," In Proc. of International Journal of Computer Science Issues, Vol. 7, Issue 5 , 2010.

[9] M. Dejan, B. Zoran, I. Mirzana and V.Milan, "Improving Fault Tolerance of Distributed Multi-Agent Systems with Mobile Network-Management Agents," In Proc. of the International Multiconference on Computer Science and Information Technology, pp. 217-222, 2010.

[10] L. Zeghache and N. Badache, "Optimistic Replication Approach for Transactional Mobile Agent Fault Tolerance," In Proc. of 11th CIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2010.

[11] P. Marikkannu, J.J. Adri Jovin, T.Purusothaman, "Fault-Tolerant Adaptive Mobile Agent System using Dynamic Role based Access Control," International Journal of Computer Applications Volume 20–No.2, April 2011.

[12] K.Ramandeep, K.C. Rama and S.Rajwinder, "Integrated mechanism to Prevent Agent Blocking in Secure Mobile Agent Platform System," In Proc. of International Conference on Advances in Computer Engineering, 2010.

[13] R.Hans, R.Kaur, "Fault Tolerance Approach in Mobile Agents for Information Retrieval Applications Using Check Points," International Journal of Computer Science & Communication Networks, Vol2(3), 347-353, 2012.

[14] M.R.Lyu and T.Y.Wong, "A Progressive Fault Tolerant Mechanism in Mobile Agent Systems," Proc. 7th World Multiconference on Systemics, Cybernetics and Informatics, vol. IX, 2003, pp. 299–306.