International Journal of Hybrid Information Technology
Vol.8, No. 5 (2015), pp. 119-128
http://dx.doi.org/10.14257/ijhit.2015.8.5.13

Building Mobile Agents' Applications Fault Tolerant

Preeti and Praveena Chaturvedi

Department of Computer Science, Gurukul Kangri Vishwavidyalaya, Haridwar
[1]preetishivach2009@gmail.com, [2]praveena_c1@rediffmail.com

Abstract

A mobile agent is self-governing software that has one or more goals and can migrate from one node to another in a network. Mobile agent has received pervasive interest in both research and academia in recent years because of its support for asynchronous and disconnected operation capability in distributed system Along with these affirmative sides, the full scale adoption of mobile agent has been delayed by several fault tolerance complexities in untrustworthy network environments such as Internet. This paper proposed a framework to provide fault tolerant execution to mobile agents. Key concepts used to accomplish this goal include the transaction manager concept, checkpointing, timeout mechanism. Agent saves partial results at home server. The framework can tolerate agent failure, server failure and communication delay. It has been implemented as an add-on to mobile agent platform called Aglets. Experiments have been conducted to test the feasibility and performance of the proposal.

Keywords: *Mobile agent, fault tolerance, checkpoint, transaction manager, timeout mechanism*

1. Introduction

Mobile Agent paradigm is a promising paradigm in distributed computing and holds a potential position in both research and academia as a result of its sole capabilities like disconnected operation, autonomous, flexibility in untrustworthy network. The phrase ***agent*** refers to self-governing software that has one or more goals, which may or may not team up with other software and users [1]. It is considered mobile when it moves from one node to another. Mobile agent is the encapsulation of program code, data and its execution state. Typical applications of mobile agents are; Remote Information Retrieval, Network Management, Mobile Computing, Software Testing, Active Networking and Active Documents etc [2]. In agent based systems, rather than moving data to code, code (mobile agent) migrates to data location that is located on remote node and sends results back to the owner after performing its computation on that data. This movement of computation or information to remote node makes mobile agents more prone to failures and security attacks [3]. To support the mobile agents in various applications areas, its execution should be fault tolerant and these areas should be considered in depth, particularly for applications that use agents in open environment such as Internet [4].

While executing on a platform, the agent may fail due to some programming errors or server crash [5]. Various fault tolerance schemes have been devised for mobile agents and these schemes can be put into two broad categories replication scheme [6, 7 and 8] and checkpointing scheme [9, 10 and 19]. In ***replication scheme***, replicated copies of the agent execute in parallel on some sites. Replication is not cost effective technique if the agent size is very large. In ***checkpointing scheme***, there is only one site to which agent is forwarded and the agent saves its state on stable storage as checkpoint. If failure occurs, the agent can be recovered from the latest checkpoint state. The two preferred properties of the mobile agent execution are; exactly-once and non-blocking [27]. Exactly-once property guarantees that in many applications an agent must be executed only once on

ISSN: 1738-9968 IJHIT

each site of its itinerary. Second property of execution is non-blocking, which guarantees that the execution does not get blocked for any cause. Replication is used to serve this purpose. If one copy of the agent fails the other replicated copies of the agent executed in parallel. So the working of the application doesn't get blocked.

Most of the researches in this area consider only search based applications of mobile agents. But there are various write applications that require exactly once execution of the mobile agent on each site. In this paper, we extend the proposal of [30] for fault tolerant execution of mobile agent. The idea is based on transaction manager and timeout concept. Checkpointing scheme is used to recover from failure of mobile agent. The agent saves partial results at home server. The server after which the agent saves results, depend on the size of the itinerary and is chosen at the time of agent creation.

Rest of the paper is organized as follows; next section discusses the problem of failures in mobile agent. Related work of this research area is given in section 3. In section 4 we describe the proposed approach along with handling various failures in section 5 then sections 6 includes experimental evaluation of the proposed fault tolerance approach and after that in section 7, we compare proposed approach with existing approaches based on some parameters followed by conclusion in section 8.

2. Problem of Failures in Mobile Agent

Before involving into the proposed mechanism for fault tolerance let us take a brief overview of the problem of failures in mobile agent paradigm. As known to all, an agent has to visit several sites to perform its designated task. A set of actions has to be performed on a single site. The basic idea of most of the protocols is to store the agent's state on stable storage. So that if a failure occurs, the agent can be recovered from this saved state. It is also important that all access to resources is performed within a step transaction [19]. If the execution of a step terminates, all the changes in resources' states have to be undone. For example, in e-commerce application, suppose an agent buys something on behalf of its owner. Agent has to credit some amount of money to the seller's account. But after crediting that amount, the agent fails. The owner of the agent somehow detects the failure and resend the agent and it will buy and pay for that item again. This is not a desirable situation. So if the agent fails after updating server or resource. These changes have to be undone.

An agent can simply fail by errors of networks or hosts or may be due to some programming errors that may lead to partial or complete loss of the agent. One more area that should also be concerned about is to differentiate between lost and delayed agents. The failure of an agent means a failure of a transaction. Whereas delayed agents are those agents that stuck in network congestion.

3. Related Work

The existing techniques to recover the failed agent includes following. Linda and Nadjib [11] proposal is based on checkpointing, chain control and message passing. Hans and Kaur [12] proposed an approach to tackle the problem of server crash by creating clone of original agent. But this approach is not suitable for write applications. Hogler *et al.* [14] developed mechanism that uses logger agent and actual agent. This mechanism is feasible and beneficial as its analytical study shows. Budi *et al.* [15] uses exception handling and structured agent coordination to suffer from system faults. It is fast and effective but violates non-blocking property of agent execution.

The proposal of Ahmed *et al.* [16] is based on replication technique. Agents in a group communicate to each other to provide reliable service. Leader of the group perform computation and put out results to other members of the group. This approach ensures both the basic property of agent execution but leader election and group management are major overheads of this technique. Marikannu *et al.* [17] proposed role sharing concept. If

an agent fails, other agents share its role. There is Role generator component to assign the roles to the agents.

Pears *et al.* [18] proposed two exception handler mechanism viz. mobile timeout design and mobile shadow design. In this approach it is not an easy task to select a perfect timeout period because of mismatch in processors speed. Dhanalakshmi and Kadhar [22] proposed a mechanism in which agent saves its results on home server after performing its computation on each server. It protects agent from malicious host or malicious agent, it means no requirement to add mechanism to secure and recover different part of the agent. This approach uses less CPU utilization, memory usage and time.

Khokhar *et al.* [24] proposed antecedence graph based approach for fault tolerance in multi-agent system. Recovery process is based on message logging. Summiya *et al.* [25] proposed a fault tolerance algorithm analogous to sliding window protocol. Agents in this approach work in collaborative manner to provide the fault free behaviour of the agent.

Isong and Ekabua [26] proposed hierarchical approach for fault tolerance mobile agent systems. Author stated that there are three phases of each fault tolerance framework; Monitor/Detection of failure, Action Planning for recovery and Execution of action and then continue service. Hans and Kaur [27] developed a mechanism that uses checkpoint based recovery method. But this is not suitable for write applications and violates exactly-once property of mobile agent execution if the agent delayed in the network, due to congestion.

Yusuf and Zaman [28] compares various fault tolerant approaches based on some defined parameters such as type of fault, management type and coordination between agents is direct or indirect.

4. Proposed Approach

In our approach an agent execution starts from home server, on which the agent is created and dispatched. The agent migrates from one server to next server. Agent saves partial results at home server. It is decided at the time of agent creation, and depends on the size of the agent itinerary that after how many servers the agent saves partial result at the home server. We call these servers checkpoint server. Checkpoint server should be chosen carefully such that overhead should not be too high that a great percentage of computation time is wasted in saving checkpoint and not too far so that agent has to rollback a huge computation if failure occurs. We have assumed the itinerary size is 16 and partial results are saved at home server after each fourth server. Fig. 1 below shows the execution process of the agent on servers. The process of execution along with system model is described in detail in next section. Table 1 summarizes the notations used throughout the paper.

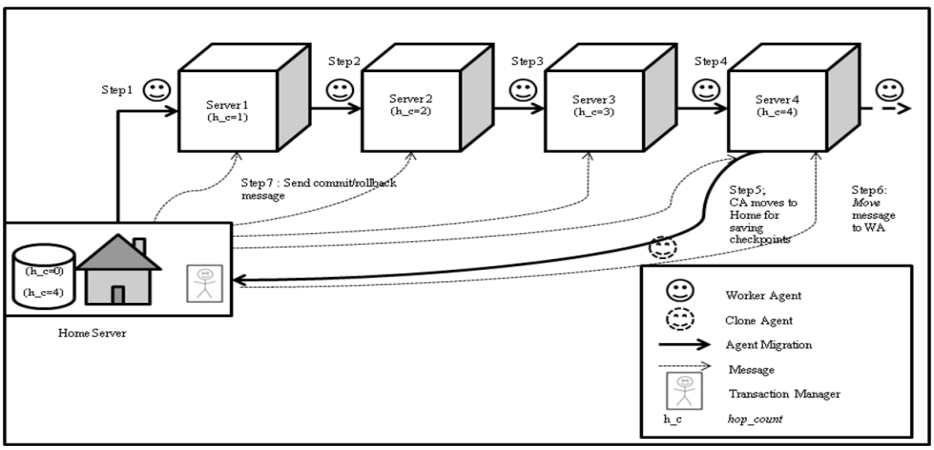

Figure 1. Mobile Agent Execution Model

Table 1. Notations Used

Symbol	Definition
TM	Transaction Manager
WA	Worker Agent
CA	Clone Agent
checkpoint time	Expected time taken by the WA for performing its computation on the servers between two concurrent checkpoints
creation time	Time of agent creation
agent message(move, ID)	TM grants permission WA to move (having identifier ID)
message(commit, ID)	TM send commit message to servers for committing the changes done by agent having identifier ID
message(rollback, ID)	TM send rollback message to servers for rolling back the changes done by agent having identifier ID
hop count	Incremented at every hop and used by WA for checking whether to save checkpoint at home server or not
Not avail server	Saves the address of the server that is not available
current time	Current system time
Last checkpoint time	Time when the agent saves results at home server

4.1. System Model

There are three components which constitute the proposed system architecture. Following section describes detailed working of these components.

- Transaction Manager (TM)
- Worker Agent (WA)
- Clone Agent (CA)

4.1.1. Transaction Manager

The TM is responsible for creating and dispatching the WA. We have implemented the TM as a stationary agent residing at home server. The TM also knows the identifier (ID) and the itinerary pattern of the WA. After Dispatching WA the TM also starts a timer. The timer is set to the *checkpoint time.* The TM wait for the timeout, if the CA (will be discussed later) is back within this time then the TM sends *Agent message(move, ID)* message to WA and resets the timer. The TM also sends message *message(commit, ID)* to the servers visited by the WA. After getting the message, the servers commit the changes done by the agent with that particular ID. And if the agent is not back, it means failed somewhere in the network, The TM sends message *message(rollback, ID)* to those servers and creates replicated copy of the WA. This replicated copy now resumes the execution from the last saved checkpoint. Variable *hop count* is used to know the servers to which the TM has to send commit or rollback message. It will be discussed in the next section. The fig. 2 below gives algorithmic description of TM.

[1] Dispatch WA.
[2] Start Timer and Register agent's ID.
[3] If CA is back within timeout period
[4] Retrieve *hop count* value.
[5] Send *move* message to WA and restart the timer.
[6] Send commit message *(commit, ID)* to servers visited between current and last saved *hop count. // suppose current hop count is 4 then send commit message to* 1^{st} , 2^{nd} , 3^{rd} *and* 4^{th} *servers.*
[7] If Itinerary is complete
    i. Stop.
[8] Else // *else of step [7]* Go to Step [3].
[9] Else wait for timeout period //*else of step [3]*
[10] After expiry of time period, send message *(rollback, ID).// suppose last saved hop count is 4 then send commit message to* 5^{st} , 6^{nd} , 7^{rd} *and* 8^{th} *servers.*
[11] Create replicated copy of original agent and starts from the server that has to be visited after the last *hop_count* value and go to Step [2]. //*if the latest hop_count is 4, send rollback to* 5^{th}, 6^{th}, *7th and* 8^{th} *servers send replicated copy to* 5^{th} *server.*

Figure 2. Working of Transaction Manager

4.1.2. Worker Agent

The WA moves from server to server with a designated task assigned to it by the owner. WA has a variable *hop count*, which is initialized to zero and at every hop this *hop count* is incremented by one. The variable *hop count* serves the purpose of knowing that to which servers the TM has to send commit or rollback message. And the agent also has a variable called *last_checkpoint_time*, which serves the purpose of checking the condition in which the agent stuck in network and the TM falsely detects the failure of the agent. Before migrating to the next server of its itinerary the WA first pings the next server whether it is available or not. If it is, then migrate to that server and if not, the WA saves this server's address in data structure *not_avail_server*. The WA skips this server and move to the next server of its itinerary.

There may be some communication delay in the network and the WA can't reach the particular server on time meanwhile the timer running on TM expires. The TM falsely detects the failure of the WA and sends the replicated copy of the WA to the places of failure. And at the same time original WA also reaches to that server which violates the exactly-once property of the agent execution. To tackle this problem, after arriving at each server, the WA checks if the difference between *current time* and its *last_checkpoint_time* is greater than *checkpoint_time* of the agent. If it is greater, WA calls dispose method and the replicated copy of the agent starts execution.

[1] Set *hop_count* =0, *creation_time*= time of creation
[2] Set *checkpoint_time*= expected time taken by WA for performing its computation on servers between two concurrent checkpoints.
[3] for i=1,2,3……n, where n is the size of agent's itinerary.
[4] If i^{th} server is available Do Step [4] to[15] *//ping next server*
[5] Migrate to that server.
[6] If ((*current_time-last_checkpoint_time)>checkpoint_time)*
//Agent's timer expires
[7] Call *dispose()* method on agent.
[8] Else //*else part of step [5]*
[9] Perform designated computation and retrieve results.
[10] If *hop_count*%4==0 *//check if it is checkpoint server*
[11] Create CA (clone agent).
[12] Wait for *move* message. After receiving *move* message.
[13] set i:=i+1.
[14] set *hop_count := hop_count*+1. //*end of if [10]*
[15] Go to Step [2].
[16] Else *// else of step [4]*
[17] not_avail_server := address_i.
[18] set i := i+1.
[19] Go to Step [2]

Figure 3. Working of Worker Agent

On the other hand, at checkpoint servers, the WA leaves a CA before migrating to next server. The purpose of the clone is described in the next section. Fig. 3 shown above gives algorithmic description of working of the WA.

4.1.3. Clone Agent

The CA serves the purpose of saving checkpoints at home server. An obvious question that arises here is why we use CA for saving result? This can also be done by the WA. If WA saves checkpoints on home server, there are overheads in moving to home server,

save results and then coming back for execution. But in our approach the WA only leave CA and move forward in its itinerary after getting move message from the TM. CA is responsible for saving results. The CA moves to the home server and communicate to TM. The TM takes the value of the hop_count variable and accordingly performs its actions.

5. Handling Failures: A Complete Scenario

To make clear the functioning of proposal, let us consider a scenario where user initiates a WA for fetching and updating some values on servers. The user creates the WA of itinerary size 12 and the checkpoint saved after each fourth server. The TM registers the WA and dispatches it with the *hop count* value initialized to zero. WA moves to the first server of its itinerary, fetches the values and updates according to user and increments the *hop_count* variable's value by 1 and same process is repeated up to 4^{th} servers. But before migrating to 5^{th} server the agent leaves CA on 4^{th} server, this CA moves to home server for saving results fetched up to this point and the value of *hop count*. So TM receive the values and results and then send "*move*" message to WA and reset the timer. The TM now sends the messages *(commit, ID)* to 1^{st}, 2^{nd}, 3^{rd} and 4^{th} servers. And this process is repeated. Following section describes various failures and recovery scenarios.

5.1. Agent Failure

Suppose the agent fails on 7^{th} server of its itinerary, the timer running on TM expires, TM check pervious *hop_count* value that is 4 and expected value of *hop_count* is 8. So the TM sends message *(rollback, ID)* to 5^{th}, 6^{th}, 7^{th} and 8^{th} servers. This message indicates that server has to rollback the changes done by the WA having identifier ID. But the WA is not executed on 8^{th} server so the 8^{th} server does not have to rollback any changes. After this the TM create replicated copy of the WA which now starts from the 5^{th} server.

5.2. Server Failure

Server may fail if any hardware or software fails due to shutdown. There are two cases of server failure; first is, server on which agent is executing fails and second is, next server of the agent's itinerary is not available. If a server crashes, the agent executing on that server also crashes. Same mechanism described above used to detect and recover from server failure. While in second case, if the next server has to be visited by the agent is not available then the agent gets blocked. Let us suppose that the WA is executed on 6^{th} server and about to migrate to the 7^{th} server and the 7^{th} server is not available due to some power supply or some other reasons. The WA gets blocked on 6^{th} server. To provide solution to this problem we add the ping mechanism to the WA. The WA first ping the next server to be visited, if it is available then migrate to that server otherwise save the address of that server and skip it. So the WA skip 8^{th} server and migrate to 9^{th} server, which will be the new checkpoint server. When the CA moves to home server it also carries the information about unavailable server, so that TM will not send messages to this particular server.

5.3. Communication Delay

Congestion is a common problem in the era of internet. Suppose the agent stuck in network and the timer running on Transaction Manager expires. So Transaction Manager falsely detects the failure of the agent and sends the replicated copy of the agent to the place of failure. Meanwhile the original agent also reaches to that particular server and both the agents get executed on that server. This scenario violates the exactly-once property of the mobile agent execution. So for solving this problem, the WA checks if the

difference between *current time* and its *last_checkpoint_time* is greater than *checkpoint_time* of the agent. If it is greater WA call dispose method. Otherwise the WA starts its designated task on that server.

6. Experimental Evaluation

To measure the performance of our approach, we implement it on Aglets System. The Aglets System is based on java and for our experiment Aglets SDK 2.0.2 is used. The Aglets is a system that provides an applet like programming model for agents. Basic elements of the Aglets are Aglet, Proxy, Context, Message, Future Reply and Identifier. Fundamental operations of an aglet are creation, cloning, dispatching, retracting, disposal, deactivation, activation and messaging [20].

6.1. Performance Parameter

- Total Trip Time: time taken by the agent to complete its itinerary and return to home server after performing its task.
- Failure Rate: itinerary size divided by number of failures.

6.2. Experiment 1: Effect of Failure Rate and Timeout Value on Total Trip Time

This experiment shows the effect of failure rate and timeout period on the total trip time of the mobile agent (fig. 4). The execution time of the WA on each server is approximately 1000 milliseconds and it takes 10 milliseconds by TM to send *move* message to the WA. For experimental analysis we have taken itinerary of size 10. The agent saves its checkpoint after each third server. If one server fails and the timeout value is 4000 milliseconds, the execution time of the agent is 14030 milliseconds (table 2) and as the timeout value increases the execution time also increases. Short timeout value also may lead to false detection of agent failure.

Table 2. Total Trip Time with Different Failure Rate and Timeout Value

Timeout Value/ Failure Rate	4000 (msecs)	6000 (msecs)	10000 (msecs)
.1	14030	16030	20030
.2	18030	22030	30030
.3	22030	28030	40030

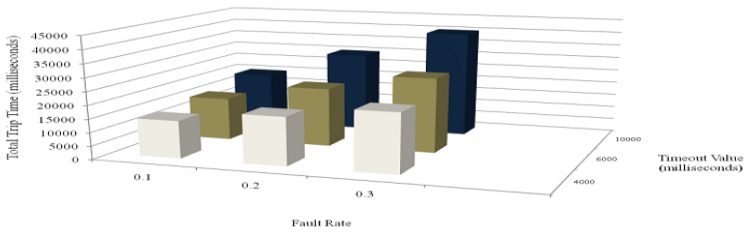

Figure 4. Total Trip Time with different Failure Rate and Timeout Value

6.3. Experiment 2: Cost of Failure-Recovery Operation

Computation loss is the amount of task which has to be redone due to the failure. We took the time to measure the computation loss. As can be seen from the graph (in fig. 5), that computation loss in our approach is much less than traditional approach (given in table 3) because our agent always starts from last saved checkpoint while in traditional approach the agent starts from the beginning.

Table 3. Computation Loss Vs Failure Rate in Proposed and Traditional Approach

Approach/ Failure Rate	Traditional Approach (msecs)	Proposed Approach (msecs)
.1	50000	2000
.2	15000	3000
.3	18000	9000
.4	20000	8000

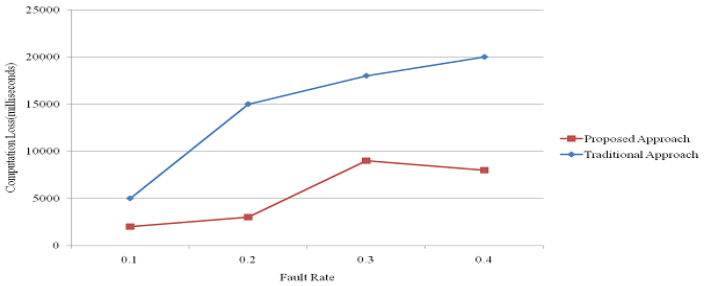

Figure 5. Computation Loss Vs Failure Rate in Proposed and Traditional Approach

Experiments prove that our proposed approach gives better results than traditional approach in terms of computation loss, because the agent does not need to start from first server, it start from the last saved checkpoint server. And usage of transaction manager also ensures that exactly-once property of mobile agent is not violated. One more thing for which we have to very careful is the selection of timeout value. Because in the network there may be a processor's speed gap, so it is difficult to choose a perfect timeout value.

7. Comparison with Existing Approaches

After reviewing the previous literature for fault tolerant mobile agent and mobile agent systems, a comparative analysis based on some common parameters studied in [28]. These parameters are:

7.1. Type of Fault

This parameter tells that what type of failure, a particular approach deal with. The failure may be agent, node/server, link failure or communication delay.

7.2. Exactly-Once Support

This parameter indicates whether the approach supports the exactly-once property of the mobile agent execution

7.3. Recovery Mechanism

Recovery mechanism may be backward (BK), forward (FR) or both. Replication is used for forward recovery in mobile agent and check pointing is used for backward recovery.

7.4. Administration

It tells whether the proposed method uses central management or not. It is an important issue from performance and cost point of view. Centralized management is easy to deal with rather than distributed management.

7.5. Agent/System Centric

If fault tolerance is achieved within the agent then the approach is said to be agent centric and if fault tolerance is achieved with the help of agent then the approach is said to be the system centric approach.
Based on these parameters, the following table 3 depicts comparative analysis of proposed approach with the existing ones.

Table 4. Comparison with Previous Approaches

Technique / Parameter	Fault	Exactly-Once Support	Recovery Mechanism	Administration	Agent/System Centric
[18]	Agent, Node	No	Both	No	Agent
[27]	Agent, Node	No	BK	No	Agent
[16]	Agent	Yes	FR	No	Agent
[29]	Agent, Node, Network	No	BK	No	System
[21]	Agent	Yes	FR	No	Agent
Proposed Approach	Agent, Node, Communication Delay	Yes	BK	Yes	Agent

8. Conclusion

Fault tolerance is vital property for execution of mobile agent based applications. In this work, a model is proposed for mobile agent fault tolerance that takes into consideration the execution properties of the agent and requirements of open environment and mobile agent execution and their applications. The proposal is based on transaction manager concept and timeout mechanism. The transaction manager performs the task of committing or rolling back the changes done by the mobile agent. Timeout mechanism is used to detect the agent failure and ping mechanism is used to check the next server availability. Proposed approach also does not suffer from overhead occurs in saving results on home server because this task is done by the clone agent. Computation loss in our approach is less due to checkpoints. Because the agent does not have to start from the scratch, if a failure occurs. Results and analysis also reflect the improvement over traditional and existing approaches.

References

[1] L. Benachenhou and S. Pierre, "Protection of a Mobile Agent with a Reference Clone", In Computer Communication (29) Elsevier, pp. 268-278, **(2006)**.
[2] I. Satoh, Mobile Agents.
[3] S. Srivastava and G. C. Nandi, "Self-reliant Mobile Code: a New Direction of Agent Security", In Journal of Network and Computer Applications (37) Elsevier, pp. 62-75, **(2014)**.
[4] F. M. A. Silva and R. J. A. Macedo, "Reliability Requirements in Mobile Agent Systems", In Proceedings of the Second Workshop on Tests and Fault Tolerance **(2000)** 15th-16th July; Curitiba, Brazil.
[5] T. Park, "Performance of K-Fault Tolerant Checkpointing for Mobile Agents", In Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications (JSAT), December Edition, pp. 16-21, **(2012).**
[6] M. Strasser and K. Rothermel, "Reliability Concepts for Mobile Agents. International Journal of Cooperative Information Systems, vol. 7, no. 4, pp. 355-382, **(1998)**.
[7] S. Pleisch and A. Schiper, "FATOMAS - A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach", In Proceeding of the International Conference on Dependable Systems and Networks **(2001)**, pp. 215-224.

[8] T. Park, I. Byun, H. Kim and H.Y. Yeom, "The Performance of Checkpointing and Replication Schemes for Fault Tolerant Mobile Agent Systems", In Proceeding of the 21st Symposium on Reliable Distributed Systems, **(2002)**, pp. 256-261.
[9] T. Park and J. Youn, "The K-Fault-Tolerant Checkpointing Scheme for the Reliable Mobile Agent System", In Proceeding of the 5th International Conference on Parallel and Distributed Computing: Applications and Technologies, **(2004)**, pp. 577-581.
[10] J. Yang, J. Cao and W. Wu, "CIC: An Integrated Approach to Checkpointing in Mobile Agent Systems", In Proceeding of the 2nd International Conference Semantics, Knowledge and Grid, **(2006)**, pp. 1-4.
[11] J. Linda and B. Nadjib, "Optimistic Replication Approach for Transactional Mobile Agent Fault Tolerance", In 11th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, **(2010)**.
[12] R. Hans and R. Kaur, "Novel Dynamic Shadow Approach for Fault Tolerance in Mobile Agent Systems", In Proceeding of 6th International Conference on Signal Processing Communication Systems, Publication IEEE Conference, **(2012)**.
[13] M. Strasser and K. Rothermel, "Reliability Concepts for Mobile Agents. In International Journal of Cooperative Information System", vol. 7, no. 4, **(1998)**, pp. 355-382.
[14] P. Holger, P. Stefan and G. Claus, "FANTOMAS: Fault Tolerance for Mobile Agents in Clusters", In Proceedings of the 15 IPDPS 2000 Workshops on Parallel And Distributed Processing, **(2000)** May 01-05, pp. 1236-1247.
[15] A. Budi, I. Alexei. R. Alexander. On Using the CAMA Framework for Developing Open Mobile Fault Tolerant Agent Systems. In Proceedings of the International workshop on Software engineering for large-scale multi-agent systems, **(2006)** May 22-23, Shanghai, China.
[16] A. E. S. Ahmed and R. R. A. El-dayem, "$62.5 US The Modeling and Implementation of Reliable Mobile Agent Systems using Group Communication Services", International Journal of Computer Application (0975-888) vol. 48, no. 6, **(2012)** June.
[17] P. Marikkannu, J. A. Jovin and T. Purusothaman, "Fault-Tolerance Adaptive Mobile Agent System using Dynamic Role based Access Control", International Journal of Computer Application, vol. 20, no. 2, **(2011)** April.
[18] S. Pears, J. Xu and C. Boldyreff, "Mobile Agent Fault Tolerance for Information Retrieval Applications", An Exception Handling Approach, **(2003)**.
[19] M. Strasser and K. Rothermel, "System Mechanism for partial Rollback of Mobile Agent execution", In Proceeding of 20th International Conference on Distributed Computing Systems, **(2000)**, pp. 20-28.
[20] D. B. Lange and M. Oshima, "Mobile Agents with Java", The Aglet API. World Wide Web (Journal), vol. 1, no. 3, **(1998)**.
[21] K. Rothermel and M. Strasser, "A Fault-Tolerant Protocol for Providing the Exactly-Once Property of Mobile Agents", In Proceeding of 17th IEEE Symposium on Reliable Distributed System, IEEE Computer Society **(1998)**, p.100-108.
[22] K. Dhanalakshmi and G. M. Kadhar Nawaz, "Matrix Hop Mobile Agent (MHMA) System for E-Service Applications", In International Conference on Communication Technology and System Design, **(2011)**, ELSEVIER.
[23] X. C. Zhong and H. Shen, "A Reliable and Secure Connection Migration Mechanism for Mobile Agents", In proceeding of 24th International Conference on Distributed Computing Systems Workshops - W4: MDC **(2004)**, p. 548-553.
[24] M. M. Khokhar, A. Nadeem and O. M. Paracha, "An Antecedence Graph Approach for Fault Tolerance in a Multi-Agent System", In Proceedings of the 7th International Conference on Mobile Data Management **(2006)**, IEEE.
[25] K. Summiya, U. Ijaz, A. Manzoor and A. Shahid, "A Fault Tolerant Infrastructure for Mobile Agents", In International Conference on Computing Intelligence For Modeling Control and Automation, and International Conference on Intelligent Agents, Web Technologies and Internet Commerce **(2006)**, IEEE.
[26] B. E. Isong and O. O. Ekabua, "A Generic Framework for Mobile Agent's Fault Tolerance", IEEE, **(2011)**.
[27] R. Hans and R. Kaur, "Fault Tolerance Approach in Mobile Agents for Information Retrieval Applications Using Check Points", In International Journal of Computer Science & Communication Networks, vol. 2, no. 3, **(2012)**, pp. 347-353.
[28] F. Yousuf and Z. Zaman, "A Survey of Fault Tolerance Techniques in Mobile Agents and Mobile Agent Systems", In Second International Conference on Environmental and Computer Science, **(2009)**.
[29] P. Dasgupta, N. Narasimhan, L. E. Moser and P. Melliar-Smith, "MAgNET: Mobile Agents for Networked Electronic Trading", IEEE Transaction On Knowledge and Data Engineering, vol. 11, no. 4, pp. 509-525, **(1999)**, July.
[30] P. Preeti, Chaturvedi and R. Hans, "A Novel Transaction Manager Based Approach for Fault Tolerance in Mobile Agent", In proceeding of Second International Conference on Emerging Research in Computing, Information, Communication and Applications'. ERCICA Elsevier, **(2014)**; pp. 380-386, Banglore.

