Fault Tolerance Mobile Agent System Using Witness Agent in 2-Dimensional Mesh Network

Ahmad Rostami[1], Hassan Rashidi[2], Majidreza Shams Zahraie[3]

[1] Student of Faculty of Electrical, Computer, IT & Biomedical Engineering, Qazvin Islamic Azad University, Qazvin, Iran

[2] Assistant Professor of Faculty of Electrical, Computer, IT & Biomedical Engineering, Qazvin Islamic Azad University, Qazvin, Iran

[3] Student of Faculty of Electrical, Computer, IT & Biomedical Engineering, Qazvin Islamic Azad University, Qazvin, Iran

Abstract

Mobile agents are computer programs that act autonomously on behalf of a user or its owner and travel through a network of heterogeneous machines. Fault tolerance is important in their itinerary. In this paper, existent methods of fault tolerance in mobile agents are described which they are considered in linear network topology. In the methods three agents are used to fault tolerance by cooperating to each others for detecting and recovering server and agent failure. Three types of agents 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. Communication mechanism in the methods is message passing between these agents. The methods are considered in linear network. We introduce our witness agent approach for fault tolerance mobile agent systems in Two Dimensional Mesh (2D-Mesh) Network. Indeed Our approach minimizes Witness-Dependency in this network and then represents its algorithm.

Keywords: *mobile agent system, mesh network*

1. Introduction

Mobile agents are autonomous objects capable of migrating from one server to another server in a computer network[1] . Mobile agent technology has been considered for a variety of applications [2] , [3] , [4] such as systems and network management [5] , [6] , mobile computing [7] , information retrieval [8] , and e-commerce [9] .

In the through mobile agent life cycle may happen failures [10] . The failures in a mobile agent system, may lead to a partial or complete loss of the agent. So the fault tolerant mobile agent systems should be created. Failures could be detected and recovered. Many of Methods exist relative to fault tolerance in mobile agent systems that are considered any of them.

Our approach is rooted from the approaches suggested in[11] , [12] . In[11] two types of agents are distinguished. The first type performs the required computation for the user which is the actual agent. Another type is to detect and recover the actual agent that is called the witness agent. These two types of agents communicate by using a peer-to-peer messages passing mechanism. In addition to the introduction of the witness agent and the messages passing mechanism, logging the actions performed by the actual agent is needed, since when failures happen, uncommitted actions must be aborted when rollback recovery is performed[13] . Check-pointed data also is used[14] to recover the lost agent.

During performing the actual agent, the witness agents are increased by addition of the servers, meaning any witness monitors the next witness and the actual agent[12] .

Our proposed scheme decreases the number of witness agents when the mobile agent itinerates in 2D-Mesh Network. The 2D-Mesh network properties are used to decrease Witness-Dependency. The algorithm of the approach is developed and simulated in C++ language.

The rest of the paper is structured as follows: Section 2 presents the method of fault tolerance in mobile agent systems by using the witness agents and probe in linear network topology. In Section 3, Mesh Network Topology and 2D-Mesh network is considered shortly. In Section 4, the proposed scheme and its method for decreasing of the

witness-dependency is described. An algorithmic view of the new approach also is presented in Section 4 and, finally, Section 5 is identified conclusion and future work.

2. Fault Tolerance in Mobile Agent Systems by Using the Witness Agents in Linear Network Topology

Recently, fault-tolerant mobile agent execution has been very active field of research. There are different methods in the literature based on linear network. Some of them are introduced in the following below.

2.1 Related work

Many method exist for fault tolerance mobile agent systems, for example primary-backup models [15] , exactly once model [16] , fault tolerant mobile agent system based on the Agent-Dependent approach [17] , mobile agent fault tolerance by exception handling approach [18] , Fault Tolerance in Mobile Agent Systems by Cooperating the Witness Agents [11] and etc. The last approach is explained in the next sub-section.

2.2 System Architecture and Protocol Design

In [11] , in order to detect the failures of an actual agent as well as recover the failed actual agent, another types of agent are designated, namely the witness agent, to monitor whether the actual agent is alive or dead. In addition to the witness agent, a communication mechanism between both types of agents is designed. In this design, agents are capable of sending messages to each other. This type of messages is called the *Direct Messages* which are peer-to-peer messages. Since a witness agent always lags behind the actual agent, the actual agent can assume that the witness agent is at the server that the actual agent just previously visited. Moreover, the actual agent always knows the address of the previously visited server. Therefore, the peer-to-peer message passing mechanism can be established.

There are cases that the actual agent cannot send a direct message to a witness agent for several reasons, e.g., the witness agent is on the way to the target server. Then, there should be a mailbox at each server that keeps those unattended messages. This type of messages is called the *Indirect Messages*. These indirect messages will be kept in the permanent storage of the target servers.

Every server has to log the actions performed by an agent. The logging actions are invoked by the agent. The information logged by the agent is vital for failure detection as well as recovery. Also, the hosting servers have to log which objects have been updated. This log file is required when performing the rollback recovery. Last but not the least, the lost agent due to the failure should be recovered when a server failure happens. However, an agent has its internal data which may be lost due to the failure. Therefore, we have to check-point the data of an agent as well as rollback the computation when necessary. A permanent storage to store the check-pointed data in the server is required. Moreover, messages are logged in the log of the server in order to perform rollback of executions. The overall design of the server architecture is shown in Fig. 1.

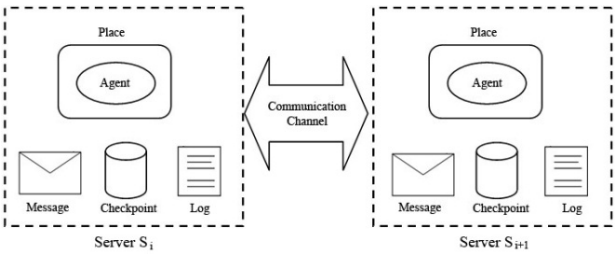

Fig. 1 The server design

The protocol is based on message passing as well as message logging to achieve failure detection. Assume that, currently, the actual agent is at server S_i while the witness agent is at server S_{i-1}. Both the actual agent and the witness agent have just arrived at S_i and S_{i-1}, respectively. The actual agent and the witness agent are labeled as α and W_{i-1} , respectively.

The actual agent α plays an active role in the designed protocol. After α has arrived at S_i, it immediately logs a message, log^{i}_{arrive} , on the permanent storage in S_i. The purpose of this message is to let the coming witness agent know that α has successfully landed on this server. Next, α informs W_{i-1} that it has arrived at S_i safely by sending a message, msg^{i}_{arrive} , to W_{i-1}.

The actual agent α performs the computations delegated by the owner on S_i. When it finishes, it immediately check-points its internal data to the permanent storage of S_i, then, it logs a message, log^{i}_{leave} , in S_i. The purpose of this message is to let the coming witness agent know that α has completed its computation and it is ready to travel to the next server S_{i+1}. In the next step, α sends W_{i-1} a message, msg^{i}_{leave} , in order to inform W_{i-1} that α is ready to leave S_i. At last, α leaves S_i and travels to S_{i+1}.

The witness agent is more passive than the actual agent in this protocol. It will not send any messages to the actual agent. Instead, it only listens to the messages coming from the actual agent. It is assumed that the witness agent, W_{i-1}, arrives at S_{i-1}. Before W_{i-1} can advance further in the network, it waits for the messages stent from α. when W_{i-1} is in S_{i-1}, it expects receiving two messages: one is

msg^{i}_{arrive} and another one is msg^{i}_{leave}. If the messages are out-of-order, msg^{i}_{leave} will be kept in the permanent storage of S_{i-1}. That is msg^{i}_{leave} is considered as unattended, and becomes an indirect message until W_{i-1} receives msg^{i}_{arrive}. When W_{i-1} has received both msg^{i}_{arrive} and msg^{i}_{leave}, it spawns a new agent called W_i. The reason of spawning a new agent instead of letting W_{i-1} migrate to S_i is that originally W_{i-1} is witnessing the availability of α. If a server failure happens just before W_{i-1} migrates to S_i, then no one can guarantee the availability of the actual agent. Note that the new witness agent knows where to go, i.e. S_i, because msg^{i}_{arrive} or msg^{i}_{leave} contains information about the location of S_i where α has just visited. Fig. 2 shows the flow of the protocol.

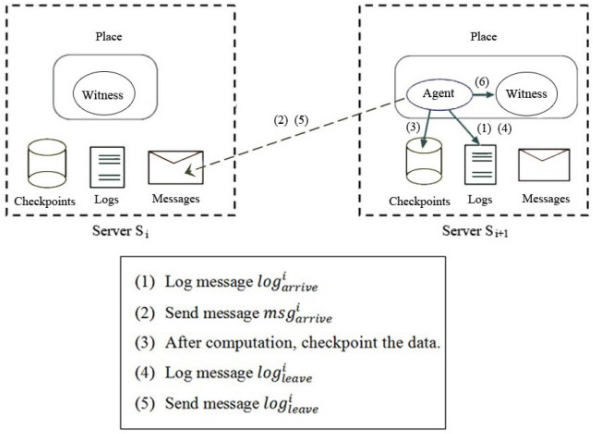

Fig. 2 Steps in the witness protocol

2.3 Creation of the Witness Agent

As it is mentioned in the previous sub-section, upon receiving msg^{i}_{leave}, W_{i-1} spawns W_i, and W_i travels to S_i. The procedure goes on until α reaches the last destination in its itinerary. The youngest (i.e., the most recently created) witness agent is witnessing the actual agent. On the other hand, the elder witness agents are neither idle nor terminated; they have another important responsibility: an earlier witness agent monitors the witness agent that is just one server closer to the actual agent in its itinerary. That is:

$$W_0 \rightarrow W_1 \rightarrow W_2 \rightarrow \ldots \rightarrow W_{i-1} \rightarrow W_i \rightarrow \alpha$$

Where "→" represents the monitoring relation. The above dependency is named the Witness- Dependency. This dependency cannot be broken. For instance, if α is in S_i, W_{i-1} is monitoring α and W_{i-2} is monitoring W_{i-1}. Assuming the following failure sequence happen: S_{i-1} crushes and then S_i crushes. Since S_{i-1} crushes, W_{i-1} is lost, hence no one monitoring α. This is not desirable. Therefore, a mechanism to monitor and to recover the failed witness agents is needed. This is achieved by the preserving the witness-dependency: the recovery of W_{i-1} can be performed by W_{i-2}, so that α can be recovered by W_{i-1}. Fig.3 shows the witness agent failure scenario.

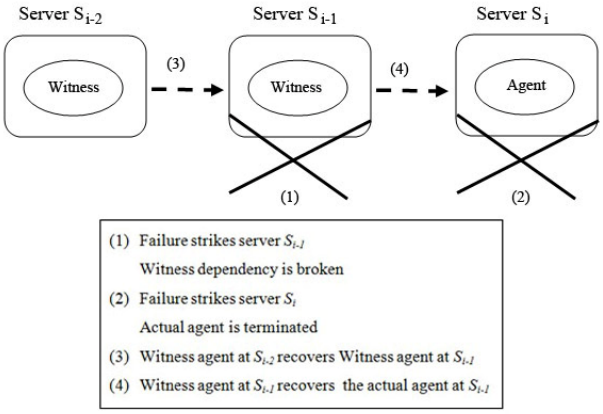

Fig. 3 The witness agent failure scenario

Note that there are other more complex scenarios, but as long as the witness-dependency is preserved, agent failure detection and recovery can always be achieved. In order to preserve the witness- dependency, the witness agents that are not monitoring the actual agent receive message from the witness agent that is monitoring it. That is, W_i sends a periodic message to W_{i-1} in order to let W_{i-1} knows that W_i is alive. This message is labeled as msg^{i}_{alive}. If W_i is dead, W_{i-1} will spawn a new witness agent, namely W_i, in order to replace the lost witness agent in S_i. When W_i arrives at S_i, it re-transmits the message msg^{i}_{alive} to W_{i-1}.

In order to handle this failure series, the owner of the actual agent can send a witness agent to the first server S_0, in the itinerary of the agent with a timeout mechanism. The effect of sending this witness agent is similar to the case when a witness agent, W_i, fails to receive msg^{i+1}_{alive}. This method can recover W_0 and the witness-dependency effectively with an appropriate timeout period.

This procedure consumes a lot of resources along the itinerary of the actual agent. Then, the number of alive-witnesses should be decreased as far as possible and minimize the witness-dependency.

3. Mesh Network

In a mesh network, the nodes are arranged into a k-dimensional lattice. Communication is allowed only between neighboring nodes; hence interior nodes communicate with 2k other nodes [19] . A k-dimensional mesh has $n_1 \times n_2 \times n_3 \times \ldots \times n_k$, where n_i is the size of i^{th} dimension. Fig. 4 illustrates a 4×4 2D-Mesh with 16 nodes.

In a mesh network, two nodes $(x_1, x_2, x_3\ldots, x_k)$ and $(x'_1, x'_2, x'_3\ldots, x'_k)$ are connected by an edge if there exist i, j such that $|x_i - x'_i|=1$ and $x_j=x'_j$ for $j\neq i$.

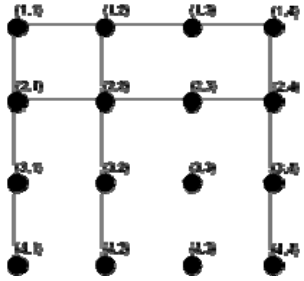

Fig. 4 A 4×4 two dimensional Mesh

4. Proposed Scheme

In our scheme the assumptions are as follows:

- The network topology is 2D-Mesh. Each agents in 2D-Mesh network are shown as a pair (i, j), where i is the number of row and j is the number of column.
- Agent itinerates dynamically in the network, i.e. it doesn't have any specified path list.
- No failure happens for the owner of the actual agent. Replication can solve the problem.

4.1 Our Approach

The main objective of our approach is to decrease the witness-dependency in 2D-Mesh. In this method, agents are in the nodes of 2D-Mesh graph. The location of each agent in the graph is shown as a pair (i, j). Assume that the owner of the actual agent hold an array, *W*, for the location of the witness agent. The *W* is empty at first and is filled during the actual agent's itinerary with the indices number of each witness. When the actual agent travels to a new server in the 2D-mesh network, it sends a message to owner after computing and check-pointing the data. This message contains the node indices which actual agent currently resides on it. This type of message is denoted as $msg^{i,j}_{NewWitnessNumber}$. This message is sent when computing and check-pointing of data are finished and the message log^{i}_{leave} is logged, because of assuring that the agent has finished its task and data stored on the stable storage. When the owner receives the message, it checks the new witness indices (i.e. new witness location in the 2D-Mesh is identified by using the number of row and column) with the existing elements in the witnesses array from the last to the first for finding its neighbor[1]. For example assume a condition which there is a node that W[e] is its indicator and this node is adjacent to the new witness; the owner sends a message to the witness (No new witness) to points to the new witness. This type of message is denoted as $msg^{i,j}_{Point NewWitness}$.

Fig. 5 shows the movement path of an actual agent from the position of (1,1) to (3,2).

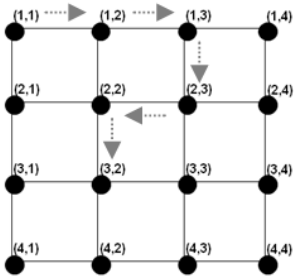

Fig. 5 The movement of the actual agent

As it can be seen, if the adjacent of nodes are not considered, then the witness-dependency is same as linear method. In this case the witness-dependency is as below:

$$W_{1,1} \rightarrow W_{1,2} \rightarrow W_{1,3} \rightarrow W_{2,3} \rightarrow W_{2,2} \rightarrow W_{3,2} \rightarrow \ldots \rightarrow \alpha$$

If the adjacent of nodes and the properties of 2D-Mesh are considered for decreasing the witness-dependency in the Fig. 5, the witness-dependency is decreased due to $W_{1,2}$ and $W_{2,2}$ which are adjacent to each other. In this case, the witness-dependency is as below:

$$W_{1,1} \rightarrow W_{1,2} \rightarrow W_{2,2} \rightarrow W_{3,2} \rightarrow \ldots \rightarrow \alpha$$

Fig. 6 shows the decreased witness-dependency on the graph.

1 In the most case that is considered, the new witness agent was neighbor with the last elements of array.

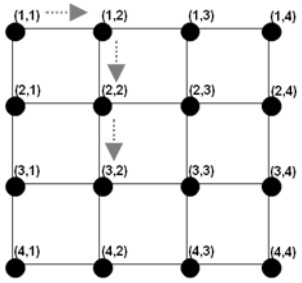

Fig. 6 The decreased witness-dependency

The fault tolerance of sending and receiving of the messages are considered in [11] . Hence, the messages $msg^{i,j}_{NewWitnessNumber}$ and $msg^{i,j}_{Point\ NewWitness}$ are not lost eventually, they are like msg^{i}_{alive} and they can be recovered as same as recovery message methods which are presented in [11] .

4.2 The Algorithm

As it is mentioned in sub-section 4.1, it is assumed that the owner of the actual agent holds an array, *W*, for the location of the witness agent, and *Index* is a pointer that points to the position of youngest witness (i.e. Index points to the last element in *W*). When actual agent α visits a new node, it sends message $msg^{i,j}_{NewWitnessNumber}$ to owner after computing and check-pointing the data. This message contains the node indices of new witness agent. For example, assume a condition that pair (p, q) is the new witness indices. These indices compare with the existing elements of array *W*. In 2D-Mesh, two nodes (x_i, x_j) and (x'_i, x'_j) are connected by an edge, if $|x_i - x'_i|=1$ and $x_j=x'_j$ or $|x_j - x'_j|=1$ and $x_i=x'_i$. Hence pair (p, q) compares with the value of *W[Index-1]*[1]. If they are adjacent then owner send $msg^{i,j}_{Point\ NewWitness}$ to the witness that *W[Index-1]* is its indicator in the array *W* for pointing to the new witness agent. Let the value of *W[Index-1]* is $W_{i,j}$. In this case if each of the below conditions is true then they are adjacent:

$$|i-p|=1 \text{ and } j=q$$

Or

$$|j-q|=1 \text{ and } i=p$$

If the above conditions are not true then Index decreases by one (i.e. Index=index-1) and then pair (p, q) compares with another element of array W. The procedure goes on until all elements of array are compared. If none of the array elements are adjacent with new witness, then the location of new witness is added to the end of array list. *Index* is succeeded by one (i.e. Index=index+1). The pseudo code of the algorithm is shown in Fig.7.

```
Procedure Main ()
{
      Let n=N×M is the number of nodes;   /* N is the
            number of rows and M is the number of
            Columns */
      Let W = new array[n]; /* W is empty at first */
      Index=W;  \* Initiate Index */
      For each visited node do
      W_{p,q}        =msg^{i,j}_{NewWitnessNumber};

      Optimize_Witness_Dependency (W_{p,q},Index);

}

Procedure Optimize_Witness_Dependency (W_{p,q},Index)
{
      For  e = Index-1 to 1 do
      {
            W_{i,j}= W[e];
            If (|i-p|=1 and j=q) or (|j-q|=1 and i=p) then
                  {
                        W [e+1] = W_{p,q}
                        Index = e+1
                        Exit procedure;
                  }/* End if */
      }/* End For */

      Index = Index + 1
      W [Index] = W_{p,q}
}
```

Fig.7 The pseudo code of the algorithm

5. Conclusions and Future Work

In this paper, at first, an existing fault tolerance method in mobile agent systems in linear network topology is considered. The existing procedure consumes a lot of resources along the itinerary of the actual agent. Then, an approach to enhance fault tolerance mobile agent systems in 2D-Mesh is proposed. In this approach witness-dependency is decreased. With regard to length of witness agent deduction, the cost of witness-dependency in 2D-Mesh network is less than linear network, because the witness length is decreased. In fact, in this approach the properties of 2D-Mesh network is used (i.e. nodes adjacency) for decreasing of witness-dependency.

[1] Whereas W[Index] is adjacent with new visited node, the comparison should be started from W[Index-1]

In the Future work, the following assumptions can be considered:

- 3D-Mesh or k-Dimensional Mesh topology: Topology of the proposed algorithm is 2D-Mesh and the algorithm can be considered in 3D-Mesh or each K-Dimensional Mesh network.
- Hyper-Cube topology: The topology can be considered in Hyper-Cube Networks.
- The topology can be considered without a coordinator for holding the array. That is the array can be part of agent's data.

References

[1] H.W. Chan, T.Y. Wong, K.M. Wong, and M. R. Lyu. "Design, Implementation and Experimentation on Mobile Agent Security for Electronic Commerce Applications." Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. pp.1871- 1878.

[2] D.Chess, B. Grosof, C. Harrison, D. Levine, C. Parris ,and G.Tsudik, "Itinerant Agents for Mobile Computing," IEEE PersonalComm. Systems, vol. 2, no. 5, pp. 34-49, Oct. 1995.

[3] D. Chess, C.G. Harrison, and A. Kershenbaum, "Mobile Agents:Are They a Good Idea?" Mobile Agents and Security, G.Vigna, ed.,pp. 25-47, Springer Verlag, 1998.

[4] D.B. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents," Comm. ACM, vol. 45, no. 3, pp. 88-89, Mar. 1999.

[5] A. Bieszczad, B. Pagurek, and T.White, "Mobile Agents for Network Management," IEEE Comm. Surveys, Sept. 1998.

[6] T. Gschwind, M. Feridun, and S. Pleisch, "ADK—Building MobileAgents for Network and Systems Management from Reusable Components," Proc. First Int'l Conf. Agent Systems and Applications/ Mobile Agents (ASAMA '99), Oct. 1999.

[7] K. Takashio, G. Soeda, and H. Tokuda, "A Mobile Agent Framework for Follow-Me Applications in Ubiquitous Computing Environment," Proc. Int'l Workshop Smart Appliances and Wearable Computing (IWSAWC '01), pp. 202-207, Apr. 2001.

[8] W. Theilmann and K. Rothermel, "Optimizing the Dissemination of Mobile Agents for Distributed Information Filtering," IEEE Concurrency, pp. 53-61, Apr. 2000.

[9] P. Maes, R.H. Guttman, and A.G. Moukas, "Agents that Buy and Sell," Comm. ACM, vol. 42, no. 3, pp. 81-91,Mar. 1999.

[10] S. Pleisch, A. Schiper, "Fault-Tolerant Mobile Agent Execution," IEEE Transaction Computer, pages 209- 222, Feb. 2003.

[11] T. Y. Wong , X. Chen, M. R. Lyu, "Design And Evalution of A Fault Tolerant Mobile Agent System,"Computer Science & Engineering Depatment the Chines University of Hong Kong, Feb 2004.

[12] S. Beheshti , M. Ghiasabadi, M. Sharifnejad, A. Movaghar, "Fault Tolerant Mobile Agent Systems by using Witness Agents and Probes, "Proc. fourth Int'l Conf. Sciences of Electronic, Technologies of Information and Telecommunications (SETIT 2007- TUNISIA), March 2007.

[13] M. Strasser , K. Pothernel," System Mechanisms for Partial Rollback of Mobile Agent Execution." Proceedings of 20th International Conference on Distributed Computing Systems 2000, pp.20-28.

[14] V. F. Nicola, "Check-pointing and the Modeling of Program Execution Time." Software Fault Tolerance, M. Lyu (ed.), John Wiley & Sons, 1994, pp.167-188.

[15] N. Budhirja, K. Marzullo, F.B. Schneider, and S. Toueg, "The Primary-Backup Approach," Distributed Systems, S. Mullender, ed., second ed., pp. 199-216, Reading, Mass.: Addison-Wesley, 1993.

[16] Q. Xin, Y. Yu, Z. Qin, "Fault Tolerance Issues in Mobile Agents," Computer Science & Engineering Department Univ. of California, San Diego, June 2000.

[17] S. Pleisch, A. Schiper, "FATOMAS – A fault-Tolerant Mobile Agent System Based on the Agent-Dependent approach," IEEE Computer Society Press. To appear in proceeding of the int'l Conference on Dependable Systems and Networks, 2001.

[18] S. Pears, J. Xu, C. Boldyreff, "Mobile Agent Fault Tolerance for Information Retrieval Application: An Exception Handling Approach," In proceeding of the Sixth IEEE International Symposium on Autunomous Decentralized systems (ISADS'03), 2003.

[19] M. J. Quinn, Designing Efficient Algorithms for Parallel Computers, University of New Hampshire: McGraw-Hill, 1976.