Fault Tolerance Mobile Agent System Using Witness Agent in Mesh Networks

Ahmad Rostami, Hassan Rashidi

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 networks 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 Mesh Networks. Indeed Our approach minimizes Witness-Dependency in this network and then represents its algorithm.

Keywords—Fault Tolerance, Mesh Networks, Mobile Agent, Witness Agent.

I. 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 Mesh Networks. The Mesh networks 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 Mesh networks 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.

II. 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.

A. 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.

B. 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](image)

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 $\alpha$ and $W_{i-1}$, respectively.

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

The actual agent $\alpha$ 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 $\text{log}^\text{leave}_i$ in $S_i$. The purpose of this message is to let the coming witness agent know that $\alpha$ has completed its computation and it is ready to travel to the next server $S_{i+1}$. In the next step, $\alpha$ sends $W_{i-1}$ a message $\text{msg}^\text{leave}_{i-1}$ in order to inform $W_{i-1}$ that $\alpha$ is ready to leave $S_i$. At last, $\alpha$ 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 $W_i$. Before $W_i$ can advance further in the network, it waits for the messages sent from $\alpha$. When $W_i$ is in $S_{i-1}$, it expects receiving two messages: one is $\text{msg}^\text{leave}_{i-1}$ and another one is $\text{msg}^\text{arrive}_{i-1}$. If the messages are out-of-order, $\text{msg}^\text{leave}_{i-1}$ will be kept in the permanent storage of $S_{i-1}$. That is $\text{msg}^\text{leave}_{i-1}$ is considered as unattended, and becomes an indirect message until $W_i$ receives $\text{msg}^\text{arrive}_{i-1}$. When $W_i$ has received both $\text{msg}^\text{arrive}_{i-1}$ and $\text{msg}^\text{leave}_{i-1}$, it spawns a new agent called $W_i$.

As it is mentioned in the previous sub-section, upon receiving $\text{msg}^\text{arrive}_{i-1}$, $W_{i-1}$ spawns $W_i$, and $W_i$ travels to $S_i$. The procedure goes on until $\alpha$ 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 “$\rightarrow$” represents the monitoring relation. The above dependency is named the Witness- Dependency. This dependency cannot be broken. For instance, if $\alpha$ is in $S_i$, $W_{i-1}$ is monitoring $\alpha$ and $W_{i+1}$ is monitoring $W_{i-1}$. Assuming the following failure sequence happen: $S_{i-1}$ crushes and then $S_i$.

![Fig. 2 Steps in the witness protocol](image)
crashes. Since $S_{i,i}$ crashes, $W_{i,i}$ is lost, hence no one monitoring $a$. 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 $W_{i,i}$ can be performed by $W_{i,2}$, so that $a$ can be recovered by $W_{i,3}$.

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 $W_{i,j}$ in order to let $W_{i,j}$ knows that $W_i$ is alive. This message is labeled as $msg_{alive}$. If $W_i$ is dead, $W_{i,j}$ will spawn a new witness agent, namely $W_n$, in order to replace the lost witness agent in $S_i$. When $W_i$ arrives at $S_i$, it re-transmits the message $msg_{alive}$ 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, $W_n$, fails to receive $msg_{alive}$. 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.

### III. MESH NETWORK

In a mesh network, the nodes are arranged into an n-dimensional lattice. Communication is allowed only between neighboring nodes; hence interior nodes communicate with 2n other nodes 0. An n-dimensional mesh has $n_1 \times n_2 \times n_3 \times \ldots \times n_n$ node, where $n_i$ is the size of $i^{th}$ dimension.

In the n-dimensional mesh network, two nodes ($v_1$, $v_2$, $v_3$, ..., $v_n$) and ($v_1'$, $v_2'$, $v_3'$, ..., $v_n'$) are connected by an edge if there exist $i, j$ such that $|v_i - v_i'| = 1$ and $v_j = v_j'$ for all $j \neq i$.

### IV. PROPOSED SCHEME

In our scheme the assumptions are as follows:

- The network topology is mesh network. Each agent in the mesh network is shown as an n-tuple ($x_1$, $x_2$, $x_3$, ..., $x_n$) where $x_1$ is the number of first dimension, $x_2$ is the number of second dimension, ... and finally, $x_n$ is the number of $n^{th}$ dimension.
- 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.

#### A. Our Approach

The main objective of our approach is to decrease the witness-dependency in the n-dimensional mesh networks. In this method, agents are in the nodes of mesh graph. The location of each agent in the graph is shown as an n-tuple ($x_1$, $x_2$, $x_3$, ..., $x_n$). Assume that the owner of the actual agent hold an array, $W_i$ for the location of the witness agent (i.e. through indices number). 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 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 $msg_{NewWitnessNumber}$. This message is sent when computing and check-pointing of data are finished and the message $log_{leave}$ is logged. When the owner receives the message, it checks the new witness indices (i.e. new witness location in the mesh network which is identified by an n-tuple ($x_1$, $x_2$, $x_3$, ..., $x_n$)) with the existing elements in the witnesses array for finding its neighbor. 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 $msg_{PointNewWitness}$.

For example assume that the actual agent itinerates in a 3-dimensional mesh network. It starts itinerary from position (2, 4, 2) to (1, 3, 2). Assume the movement path be as follows:

$$(2, 4, 2) \rightarrow (2, 4, 1) \rightarrow (2, 3, 1) \rightarrow (2, 2, 1) \rightarrow (2, 2, 2) \rightarrow (2, 3, 2) \rightarrow (1, 3, 2) \rightarrow \alpha$$

In this case the witness-dependency is as below:

$$W_{2,4,2} \rightarrow W_{2,4,1} \rightarrow W_{2,3,1} \rightarrow W_{2,2,1} \rightarrow W_{2,2,2} \rightarrow W_{2,3,2} \rightarrow W_{1,3,2} \rightarrow \alpha$$

As it can be seen, if the adjacent of nodes are not considered, then the witness-dependency is same as linear method. If the adjacent of nodes and the properties of mesh network are considered, the witness-dependency is decreased due to $W_{2,4,2}$ and $W_{2,3,2}$ which are adjacent to each other. In this case, the witness-dependency is as below:

$$W_{2,4,2} \rightarrow W_{2,3,2} \rightarrow W_{1,3,2} \rightarrow \alpha$$

Fig. 3 (a) and (b) show the movement path of the actual agent before decreasing witness-dependency and the decreased witness-dependency respectively.
Fig. 3 (b) shows that the number of alive-witnesses is decreased as far as possible and the witness-dependency is minimized. Hence the consumption of the resources along the itinerary of the actual agent decreases.

B. The Algorithm

As it is mentioned in previous sub-section, 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 $\alpha$ visits a new node, it sends message $msg_{NewWitnessNumber}$ to owner. This message contains the node indices of new witness agent. For example, assume a condition that an $n$-tuple $(x_1, x_2, x_3, …, x_n)$ is the new witness indices. These indices compare with the existing elements of array $W$ from the last to the first\(^2\). Hence $n$-tuple $(x_1, x_2, x_3, …, x_n)$ compares with the value of $W[Index-1]$\(^2\). Let the value of $W[Index-1]$ is $y_1, y_2, y_3, …, y_n$. If they are adjacent, then owner sends a $msg_{PointNewWitness}$ to the witness that $W[Index-1]$ is its indicator in the array $W$ for pointing to the new witness agent (i.e. the witness agent which is most recently visited by actual agent). If each of the below conditions is true then they are adjacent:

- $\left| x_1 - y_1 \right| = 1$ and $x_j = y_j$ for all $j \neq 1$, $1 \leq j \leq n$
- $\left| x_2 - y_2 \right| = 1$ and $x_j = y_j$ for all $j \neq 2$, $1 \leq j \leq n$
- ...
- $\left| x_n - y_n \right| = 1$ and $x_j = y_j$ for all $j \neq n$, $1 \leq j \leq n$

If the above conditions are not true then $Index$ decreases by one (i.e. $Index= Index-1$) and then $n$-tuple $(x_1, x_2, x_3, …, x_n)$ compares with another element of array $W$ (i.e. $W[Index-2]$). 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 (i.e. $W_{x_1, x_2, x_3, …, x_n}$) is added to the end of array list (i.e. $W[Index+1]$). $Index$ variable is succeeded by one (i.e. $Index= Index+1$). The pseudo code of the algorithm is shown in Fig.4.

\(^2\) In the most case that is considered, the new witness is neighbor with the last elements of array.

\(^3\) Whereas $W[Index]$ is adjacent with new visited node, the comparison should be started from $W[Index-1]$.

V. 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 mesh networks is proposed. In this approach witness-dependency is decreased. With regard to length of witness agent deduction, the cost of witness-dependency in mesh networks is less than linear network, because the witness length is decreased. In fact, in this approach the properties of mesh networks is used (i.e. nodes adjacency) for decreasing of witness-dependency.

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

- Hyper-Cube topology: The topology can be considered in Hyper-Cube Networks and other topologies.
- The topology can be considered without a coordinator for holding the array. The array can be part of agent’s data.
REFERENCES


