A Model for Mobile Code Computing Paradigms in Computer Networks

Hodjat Hamidi*
Department of Industrial Engineering, K. N. Toosi University of Technology, Tehran, Iran
h_hamidi@kntu.ac.ir
Maryam Parvini
Department of Industrial Engineering, K. N. Toosi University of Technology, Tehran, Iran
parvini@mail.kntu.ac.ir

Received: 22/Feb/2016 Revised: 18/Feb/2017 Accepted: 08/Mar/2017

Abstract

This paper presents a reliable model for mobile codes in distributed networks, which represents reliable mobile agent execution. The model ensures non-blocking mobile agent execution and forces the once property without relying on correct fault detection. A mobile agent execution is blocking if a fault of agent prevents the agent from continuing in its execution. The once problem is related to non-blocking in the sense that solutions to the latter may lead to multiple executions of the mobile agent. A solution to reliable mobile agent execution needs to ensure both the non-blocking and once properties. The analytical results show new theoretical perceptions into the statistical behaviors of mobile agents and provide useful tools for executing mobile agents in networks. The results show that agents' behavior is influenced by places' characteristics and the agents' behavior can be managed to network. In this paper, we analyzed the average time consuming of mobile agents between two places. The approach, Fault-Tolerant approach for mobile codes offers a user-transparent fault tolerance which can be selected by the user for every single application given to the environment. Thereby, the user can decide for every application weather it has to be treated fault-tolerant or not. We proposed a reliable execution model of mobile codes and analyzed the life expectancy, including the average time consuming of mobile agents between two places, the average number of places agents will visit, and the agents' life expectancy.

Keywords: Mobile Computing; Mobile Code; Computer Network; Computing Paradigms.

1. Introduction

In view of the deficiencies of the client/server paradigm, the mobile code paradigm has been developed as an alternative approach for distributed application design. In the client/server paradigm, programs cannot move across different places and must run on the places they reside on. The mobile code paradigm, on the other hand, allows programs to be transferred among and executed on different computers. By allowing code to move between places, programs can interact on the same computer instead of over the network. Therefore, communication cost can be reduced. Besides, mobile agent [1-2] programs can be designed to work on behalf of users autonomously. This autonomy allows users to delegate their tasks to the mobile agents, and not to stay continuously in front of the computer terminal. The promises of the mobile code paradigm bring about active research in its realization. Most researchers, however, agree that security concerns are a hurdle [3]. In this paper, we investigate these concerns. A mobile agent is a software program which migrates from a site to another site to perform tasks assigned by a user. For the mobile agent system to support the agents in various application areas, the issues regarding the reliable agent execution, as well as the compatibility between two different agent systems or the secure agent migration, have been considered. Some of the proposed schemes are either replicating the agents [4-5] or checkpointing the agents [6-7]. For a single agent environment without considering inter-agent communication, the performance of the replication scheme and the checkpointing scheme is compared in [8] and [9]. In the area of mobile agents, only few work can be found relating to fault tolerance. Most of them refer to special agent systems or cover only some special aspects relating to mobile agents, e. g. the communication subsystem. Nevertheless, most people working with mobile agents consider fault tolerance to be an important issue [10]. Cluster, and therefore parallel applications running on them, are very susceptible for failures of components of the cluster. However, programmers and users of distributed applications are interested in their algorithms and solutions. They expect fault tolerance as a service from the underlying run time system. These considerations show the necessity for a design, which enables user-transparent fault tolerance in agent environments. Current agent systems, and also the underlying operating systems, provide this feature only insufficiently, if at all. In this paper we introduce an approach for such a design. It can be applied to different agent systems, if they fulfill certain requirements as discussed below. The approach, Fault-Tolerant approach for mobile agents offers a user-transparent fault tolerance which can be selected by the user for every single

* Corresponding Author

application given to the environment. Thereby, the user can decide for every application weather it has to be treated fault-tolerant or not.

The paper is organized as follows: Section 2 presents the main security challenge of mobile code. In Section 3, the security modeling for the mobile agent and model failures are explained. In Section 4, the fault-tolerant execution model is introduced. The simulation result is discussed in section 5. Conclusion is given in Section 6.

2. The Main Security Challenge of Mobile Code

The main security challenge of mobile code systems lies on the protection of agents. When an agent executes on a remote host, the host is likely to have access to all the data and code carried by the agent. If by chance a host is malicious and abuses the code or data of an agent, the privacy and secrecy of the agent and its owner would be at risk.

Seven types of attack by malicious hosts [2] can be identied:

- Spying out and manipulation of code;
- Spying out and manipulation of data;
- Spying out and manipulation of control flow;
- Incorrect execution of code;
- Masquerading of the host;
- Spying out and manipulation of interaction with other agents; and
- Returning wrong results of system calls to agents

There are a number of solutions proposed to protect agents against malicious hosts [10], which can be divided into three streams:

- Establishing a closed network: limiting the set of hosts among which agents travel, such that agents travel only to hosts that are trusted.
- Agent tampering detection: using specially designed state-appraisal functions to detect whether agent states have been changed maliciously during its travel.
- Agent tampering prevention: hiding from hosts the data possessed by agents and the functions to be computed by agents, by messing up code and data of agents, or using cryptographic techniques.

Depending on the choices made on the client and server sides, the following variants of mobile code computing paradigms can be identified [11-12]:

In the Remote Evaluation (REV) paradigm, component A sends instructions specifying how to perform a service to component B. These instructions can, for instance, be expressed in Java byte code. Component B then executes the request using its own resources, and returns the result, if any, to A. Java Servers are an example of remote evaluation [13].

In the Code on Demand (CoD) paradigm, the resources are collocated with component *A*, but *A* lacks the knowledge of how to access and process these resources in order to obtain the desired result. Rather, it gets this information from component *B*. As soon as *A* has the necessary know-how (i.e., has downloaded the code from *B*), it can start executing. The mobile agent computing paradigm is an extension of the REV paradigm. Whereas the latter focuses primarily on the transfer of code, the mobile agent paradigm involves the mobility of an entire computational entity, along with its code, the state, and potentially the resources required to perform the task. As developer-transparent capturing and transfer of the execution state (i.e., runtime state, program counter, and frame stacks, if applicable) requires global state models as well as functions to externalize and internalize the agent state, only few systems support this *strong mobility* scheme. In particular, Java-based mobile agent platforms are generally unsuitable for this approach, because it is not possible to access an agent's execution stack without modifying the Java Virtual Place. Most systems thus settle for the *weak mobility* scheme where only the data state is transferred along with the code. Although it does not implicitly transport the execution state of the agent, the developer can explicitly store the execution state of the agent in its member attributes. The values of these member attributes are transported to the next place. The responsibility for handling the execution state of an agent thereby resides with the developer. In contrary to REV, mobile agents can move to a sequence of places, i.e., can make multiple hops. The mobile code paradigm is actually a collective term, applicable wherever there is mobility of code. Different classes of code mobility can be identified, whereas Ghezzi and Vigna proposed three of them, namely *remote evaluation*, *code on demand* and *mobile agent* [14-15].

In particular, the code that is to be executed for the specific task. In the mobile code paradigms (*remote evaluation*, *code on demand*, and *mobile agent*), the *know-how* moves from one side to another side regarding where the computation takes place; while in the client/server paradigm, the *know-how* is stationary on the remote (server) side. *Resources* are the input and output for the code, whereas *processor* is the abstract place that carries out and holds the state of the computation. The arrows represent the directions in which the specific item should move before the required task is carried out. Ghezzi and Vigna's classification, [15], is found to be comprehensive and representative of most existing mobile code paradigms (such as the rsh utility, Java applets and mobile agent systems), and we will base our discussion on this classification.

3. Security Modeling

There are several fault tolerance issues that need to be addressed in our approach, just as in other schemes. For example, when storage space is exceeded in data bin services, some form of queue management is implemented (much like routers discard packets under certain load conditions). One or more trusted third parties can be used for data collection activities or task agent

hosting (instead of the originating host) to allow for disconnected host operations. Timeout of task agents that must wait for results of both the computation agent and the data collection agents can be mitigated by providing time-based services that determine when agents have been unreasonably detained or diverted.

As with any multi-agent or mobile agent system, recovery from errors when messages are not delivered or when migration is not possible needs to be addressed. Failure of data bin services would require an alternative or default data storage service in the network if the host facility becomes unavailable. Failure of the original task agent, failure of one or more computation agents, and failure of data collection agents can be mitigated by such approaches as the shadow model of [14]. Other work on fault-tolerance such as [15-42] provides approaches to mitigate host failures and malicious activity. Denial of service or random alterations of the code are not preventable because the agent server has ultimate power over an agent by having access to executable code and updatable state—though such activity can be detectable. When multi-hop agents with dependent (aggregated) data are used, the ability to mask or guard the function itself is needed to protect the computation agent against smart alterations of the code. We are currently researching other means to accomplish this aspect of agent protection [16] and plan to incorporate future results in consideration of multi-hop migrations. We also do not address the ability to keep keys used by both the computation and collection agent private, though it is an important issue with planned future research along the lines of work such as [17-18].

Several types of faults can occur in agent environments. Here, we first describe a general fault model, and focus on those types, which are for one important in agent environments due to high occurrence probability, and for one have been addressed in related work only insufficiently.

- Node failures: The complete failure of a compute node implies the failure of all agent places and agents located on it. Node failures can be temporary or permanent.
- Failures of components of the agent system: Failures of agent places, or components of agent places become faulty, e. g. faulty communication units or incomplete agent directory. These faults can result in agent failures, or in reduced or wrong functionality of agents.
- Failures of mobile agents: Mobile agents can become faulty due to faulty computation, or other faults (e. g. node or network failures).
- Network failures: Failures of the entire communication network or of single links can lead to isolation of single nodes, or to network partitions.
- Falsification or loss of messages: These are usually caused by failures in the network or in the communication units of the agent systems, or the underlying operating systems. Also, faulty transmission of agents during migration belongs to this type.

Especially in the intended scenario of parallel applications, node failures and their consequences are important. Such consequences are loss of agents, and loss of node specific resources. In general, each agent has to fulfill a specific task to contribute to the parallel application, and thus, agent failures must be treated. In contrast, in applications where a large number of agents are sent out to search and process information in a network, the loss of one or several mobile agents might be acceptable [19-20].

Places, or agents can fail and recover later. A component that has failed but not yet recovered is called down; otherwise, it is up. If it is eventually permanently up, it is called good [21]. In this paper, we focus on crash failures (i.e., processes prematurely halt). Benign and malicious failures (i.e., Byzantine failures) are not discussed. A failing place causes the failure of all agent running on it. Similarly, a failing node causes all places and agents on this node to fail as well. We do not consider deterministic, repetitive programming errors (i.e., programming errors that occur on all agent replicas or places) in the code or the place as relevant failures in this Context. Finally a link failure causes the loss of the messages or agents currently in transmission on this link and may lead to network partitioning. We assume that link failures (and network partitions) are not permanent. The failure of a component (i.e., agent, place, node, or communication link) can lead to blocking in the mobile agent execution. Assume, for instance that place P_2 fails while executing a_2 (Fig. 1). While P_2 is down, the execution of the mobile agent cannot proceed, i.e., it is blocked. Blocking occurs if a single failure prevents the execution from proceeding. In contrast, and execution is non-blocking if it can proceed despite a single failure, the blocked mobile agent execution can only continue when the failed component recovers.

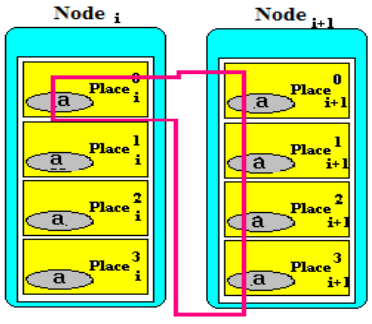


Fig. 1. the redundant places mask the place failure
(Shaded rectangles represent transactional message queues, whereas the dotted line indicates the borders of a node transaction), [2]

This requires that recovery mechanism be in place, which allows the failed component to be recovered. If no recovery mechanism exists, then the agent's state and, potentially, even its code may be lost. In the following, we assume that such a recovery mechanism exists (e.g., based on logging [22-23]. Replication prevents blocking. Instead of sending the agent to one place at the next node, agent replicas are sent to a set $node_i$ of places $P_i^0, P_i^1, \ldots$

(Fig. 1). We denote by a_i^j the agent replica of a_i executing on place P_i^j, but will omit the superscripted index if the meaning is clear from the context. Although a place may crash (i.e., $node_1$ in Fig. 1), the agent execution does not block. Indeed, P_2^1 can take over the execution of a1 and thus prevent blocking. Note that the execution at $node_0$ and $node_2$ is not replicated as the agent is under the control of the user. Moreover, the agent is only configured at the agent source and presents the results to the agent owner at the agent destination. Hence, replication is not needed at these nodes.

Despite agent replication, network partitions can still prevent the progress of the agent. Indeed, if the network is partitioned such that all places currently executing the agent at $node_i$ are in one partition and the places of $node_{i+1}$ are in another partition, the agent cannot proceed with its execution. Generally (especially in the Internet), multiple routing paths are possible for a message to arrive at its destination. Therefore, a link failure may not always lead to network partitioning. In the following, we assume that a single link failure merely partitions one place from the rest of the network .Clearly, this is a simplification, but it allows us to define blocking concisely. Indeed , in the approach presented in this article, progress in the agent execution is possible in a network partition that contains a majority of places .If no such partition exists , the execution is temporally interrupted until a majority partition is established again ,Moreover , catastrophic failures may still cause the loss of the entire agent. A failure of all places in $node_1$ (Fig. 1), for instance, is such a catastrophic. Failure (assuming no recovery mechanism is in place). As no copy of *a1* is available any more, the agent *a1* is lost and, obviously, the agent execution can no longer proceed .In other words, replication does not solve all problems. The definition of non-blocking merely addresses single failures per node as they cover most of the failures that occur in a realistic environment. In the next section, we classify the places in $node_i$ into iso-places and hetero – places according to their properties [16-17].

An agent *"a"* can commit if all or some of the surrogates commit depending on the commitment condition Com (*a*). Each agent is also realized by using the XA interface [18-20] which supports the two-phase commitment protocol. Each surrogate issues a prepare request to a server on receipt of a prepare message from the agent. If prepare is successfully performed, the surrogate sends a prepared message to the agent. Here, the surrogate is referred to as committable. Otherwise, the surrogate aborts after sending aborted to the agent. The agent receives responses from the agents after sending prepare to the surrogates. On receipt of the responses from surrogates, the agent makes a decision on commit or abort based on the commitment condition. In the atomic condition, the agent sends commit only if prepared is received from every surrogate. The agent sends abort to all committable servers if aborted is received from at least one surrogate. On receipt of abort, a committable surrogate aborts. In the at-least-one commitment condition, the agent sends commit to all committable servers only if prepared is received from at least one object server. Surrogate a_i asks the other surrogate if they had committed. Suppose the surrogate a_i is faulty before receiving prepared. Here, a_i is abort able. If the surrogate a_i is recovered, a_i unilaterally aborts (Fig.2).

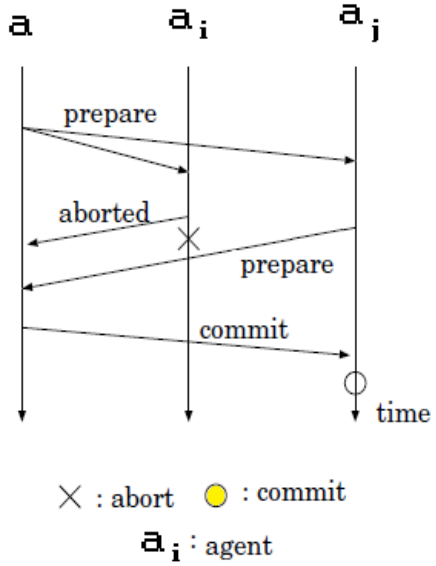


Fig. 2. Conditional commitment [4].

Now, let us consider a mobile agent travelling through *n* places on the network. Each place, and the agent itself, is modeled as an abstract node as in [17]. We consider only the standard attack phase described in [18] by malicious places. On arrival at a malicious place, the mobile agent is subject to an attack effort from the place. Because the place is modeled as a node, it is reasonable to estimate the attack effort by the number of instructions for the attack to carry out, which would be linearly increasing with time. On arrival at a non-malicious place, the effort would be constant zero. Let the agent arrive at place *i* at time T_i, for *i=1,2...n*. Then the effort of place *i* at total time / would be described by the *time-to-effort function* [1, 2]:

$$E_i(t) = k_i(t - T_i), \quad (1)$$

where *k* is a constant.

We may call the constant k_i the *coefficient of malice*. The larger the k_i, the more malicious place *i* is (k_i =0 if place *i* is non-malicious). Furthermore, let the agent stay on place *i* for an amount of time t_t, then there would be breach to the agent if and only if the following breach condition holds:

$E_i(t_i+T_i)$ > effort to next breach by place *i* (2)

i.e., $k_i t_i$ > effort to next breach by place *i*

As seen from [19-20], it is reasonable to assume exponential distribution of the effort to next breach, so we have the *probability of breach at place i*,

P (breach at place *i*) = P (breach at time t_i+T_i) (3)

= P (breach at effort $k_i t_i$)

$= 1 - exp(-vk_i t_i)$, v is a constant

$= 1 - exp(-\lambda_i t_i)$, $\lambda_i = vk_i$

We may call v the *coefficient of vulnerability* of the agent. The higher the v, the higher is the probability of breach to the agent. Therefore, the *agent security E* would be the probability of no breach at all places, i.e.

$$E = e^{-\sum_{i=1}^{n} \lambda_i t_i} \quad (4)$$

Suppose that we can estimate the coefficients of malice k_i's for places based on trust records of places, and also estimate the coefficient of vulnerability v of the agent based on testing and experiments, then we can calculate the desired time limits T_i's to achieve a certain level of security *E*. Conversely, if users specify some task must be carried out on a particular place for a fixed period of time, we can calculate the agent security *E* for the users based on the coefficients of malice and vulnerability estimates.

4. The Fault Tolerant Execution Model and Evaluation

For a large network with a large number of node and place, suppose that agents can be generated from every place on networks, provide mobile agents an execution environment. Initially, there are a pile of tasks generated in the network .Then a pile of agents, whose number is equal to that of the tasks, is generated. Each task is carried by an agent .Those agents wander among places in the network to search for their destinations. At each place, agents have local information about the error rate of each adjoin link, but they do not have global knowledge on the state of the network. The sequence of places visited by the agent compose the agent's itinerary. Agents' itineraries can be either static or dynamic. A static itinerary is entirely defined at the source and does not change during the agent traveling; whereas a dynamic itinerary is subject to modifications by the agent during its execution [21].

Since mobile agents are capable of sensing the execution environment and reacting autonomously to changes [22], a dynamic itinerary on the fly .Let P_i denote the i-th place in the itinerary and P (i) denote the set consisted by the neighbor places of P_i .The number of neighbor places in set P (i) is denoted by l_i, i.e., the connectivity degree of place P_i. Once an agent reaches a place it executes locally. After completed its execution, the agent selects a place from P (i) to move to. Suppose that there is an error rate for each candidate direction, mobile agents will prefer a route with a low error rate to shun faults. The selected place in P (i) is denoted by P_{i+1}^1. In case that a failure takes place on P_{i+1}^1, the agent is blocked and has to return to the previous place P_i. Then, it will reselect another neighbor place from P (i) and move to. The j-th selected place in P(i) is denoted P_{i+1}^1. An agent is supposed will not jump to the same neighbor place twice since in a general way a failure place will not recover in a very short time. This process will continue until the agent successfully enters a place and completes its execution there. The final visited place in P (i) is denoted by P_{i+1}.

Communication between consecutive nodes P_i and P_{i+1} is based on transactional message queues, shown as shaded rectangles in Fig. 1. At each node, a place retrieves the agent from its input queue, executes the agent, and places the resulting agent in the input queues of the next node's places as one transaction. A place P_i can only commit the distributed transaction when it is elected by the places in $node_i$, when it receives a majority of votes. Rothermel uses a 2-phase commit protocol [23] to commit the transactions, the election protocol thereby acting as a resource manager to the transaction manager. Modeling reliable mobile agent execution based on two different, interfering problems leads to a more complex solution than ours. In addition, understanding the weaknesses of such a solution is difficult and tedious. Our solution, however, is specified in terms of a *single* problem, the consensus problem, an intensively studied problem with well-understood solutions.

In Rothermel's model, the execution of the agent as well as the forwarding of the agent from node P_i to P_{i+1} run as a transaction. Our model, in contrast, clearly decouples the mechanisms that provide fault tolerance from the execution properties of the agent operations. In particular, the agent operations do not need to run as a transaction. If they do, they have their own transaction manager.

In an asynchronous distributed system, there are no bounds on transmission delays of messages or no relative process speeds. Therefore, when a mobile agent is blocked by reason of a failure in an asynchronous distributed system, the agent owner cannot correctly determine whether the agent has failed or is merely slow [24-28]. Therefore, the reliability of agents' execution is paramount for measuring the network performance. We treat this problem as a probability problem using the behavior of mobile.

Agents to build a probability estimation on the number of places an agent can visit. Let S_i denote the number of places selected by an agent in set P(i). The event indicates that the agent cannot enter the place P_i^j in set P (i), then the parameter p measures the incidence of failure in the network. The average number of selected places in set P (i), denoted by M (Si), and satisfies.

$$M(S_i) = (1-(p(1-p))^{d_i}) / (1-p+p^2))) \quad (5)$$

i=1,2… and j=1,2,…….

From Fig.3, it is easy to see that the average number of places an agent will selected in a neighbor place set is an increase function on both error rate p and the number of neighboring places d_i. Furthermore, if the time cost for passing a link approximates to a constant *k*, we have estimate the average time consumption for mobile agents entering a place in set P (i).

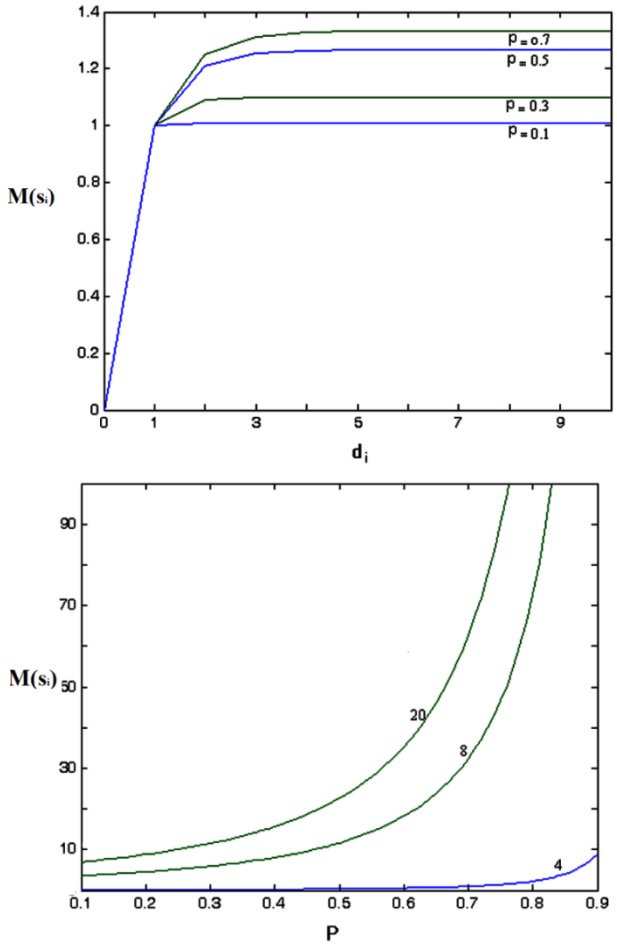
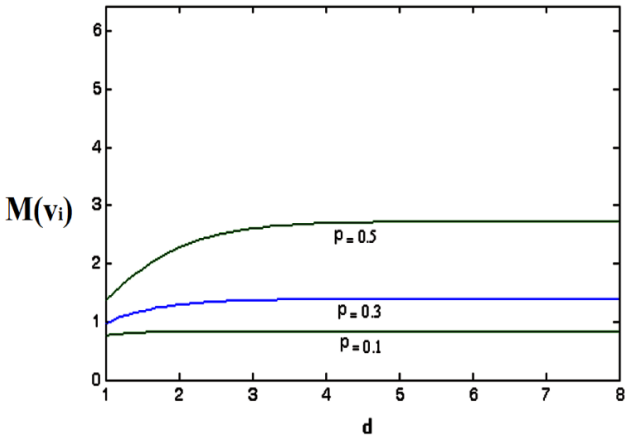


Fig. 3. The Changes of M (Si) over p and d_i

By the assumption that the time consumption for an agent passing a link is *q*, the time consumption of the period that an agent moves to a down place and returns to the previous place equals to *2q*. Hence, Agents' life expectancy satisfies.

$$M[v] = (q(1-q^d)/((1-q)(1+p))/((1-p((1-q^d)/(1-q))) \quad (6)$$



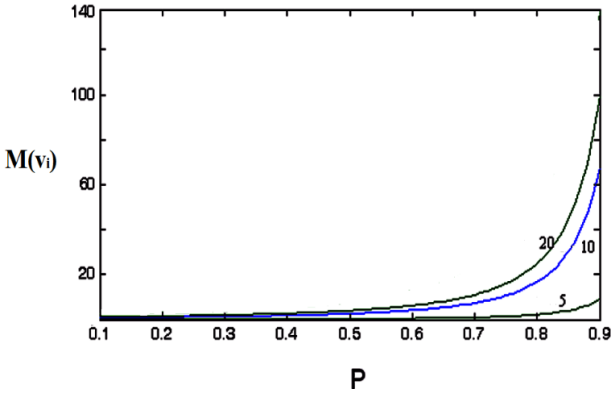


Fig. 4.The Changes of M (vi) over P and d

Fig. 4. shows the changes of agents' average life expectancy. It is easy to see that the average life expectancy is an increase function on both the error rate and the network connectivity. In particular, it is a convex function on the parameter *d* and a concave function on the parameter p.

5. Results

In this experiment, we change the number of nodes from 2 to 30 and use one mobile agent. The result is shown in Fig. 5(a). We observe that both the execution time and the energy consumption using either computing model grow as the number of nodes increases. But the execution time of the client/server model grows much faster than the mobile-agent-based model. This is because as the number of nodes increases, the server has to deal with more connections requested by the clients at the same time, which elongates the execution time. On the other hand, the mobile agent model is less influenced by the number of nodes because there are far fewer connections at one time for the mobile agent model. The figure also shows that the client/server model performs a little better than the mobile agent model from both the execution time and energy consumption perspectives. This happens when the mobile agent model needs more connections than the client/server model in order to send and receive mobile agents. It also happens when the overhead of the mobile agent surpasses the overhead of the client/ server model.

In this experiment, we fix the node number at 100 but change the number of mobile agents from 1 to 50. Without loss of generality, we assume each agent migrates the same number of nodes. We expect a constant profile from the client/server model since it is irrelevant to the number of mobile agents. We can see from Fig. 5(b) that the execution time of the mobile agent model is always less than that of the client/server model because the node number is large. Interestingly, the execution time of the mobile agent model decreases as the number of mobile agents increases and reaches the lowest point when there are five mobile agents. Then, the execution time begins to climb. This is because more mobile agents

will reduce the number of nodes each agent migrates, thus reducing the execution time. But more mobile agents also cause more connections and more overheads. As the number of mobile agents increases, the energy consumption also increases in linear and the mobile agent model actually consumes more energy when m>15.

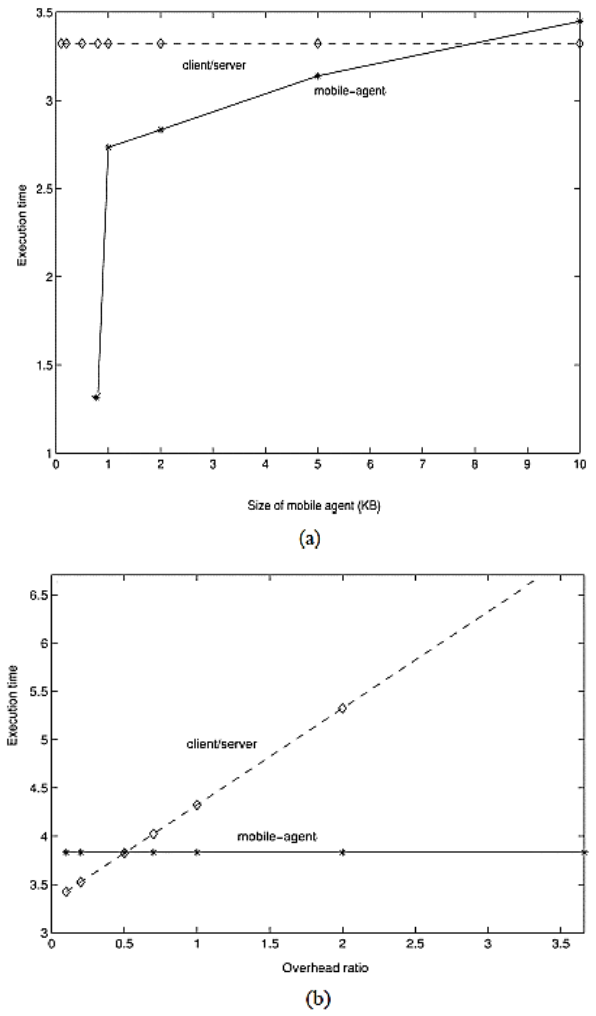


Fig. 5. (a) Effect of the mobile agent size (s). Execution time. (b) Effect of the overhead ratio and Execution time.

Access time from time when the application program starts to time when the application program ends, is measured for Agents and client-server model. Fig.6 shows the access time for number of object servers. The mobile agent's shows that Aglets classes are not loaded when an agent A arrives at an object server. Here, the agent can be performed after mobile agents are loaded. On the other hand, the mobile agent's with replication means that an agent manipulates objects in each object server where mobile agents are already loaded, i.e. the agent comes to the object server after other agents have visited on the object server. As shown in Fig.6, the client-server model is faster than the transactional agent. However, the transactional agent is faster than the client-server model if object servers are frequently manipulated, i.e. mobile agents with replication are a priori loaded.

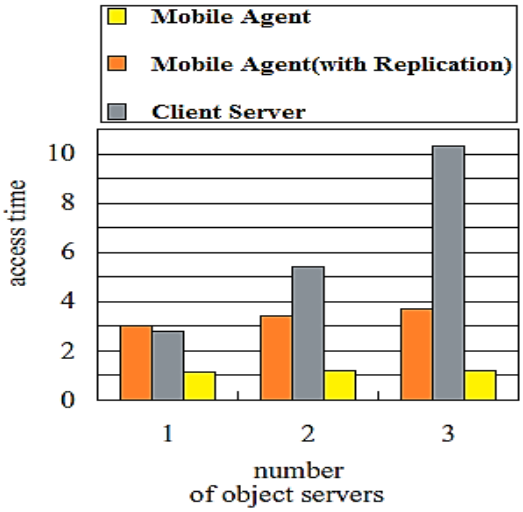


Fig. 6. Client server vs. agent.

6. Conclusion

To achieve reliable in the context of mobile codes, we first have specified reliable mobile agent execution in terms of two properties: non-blocking and once execution. Replication overcomes the blocking problem. This paper shows how the present approach to reliable mobile agent execution can be used to achieve non-blocking mobile agent execution. The use of mobile agent, however, is critical and requires reliability in regard to mobile agent failures that may lead to bad response time and hence the availability of the system may lost. In this paper, a fault tolerance technology is proposed in order that the system autonomously detect and recover the fault of the mobile agent due to a failure in a transmission link. The key idea is the use of stochastic regularities of mobile agent's behavior-all the mobile agents in the network as a whole can be stochastically characterized though a single mobile agent may act randomly. In this paper, we proposed a reliable execution model of mobile agents and analyzed the life expectancy, including the average time consuming of mobile agents between two places, the average number of places agents will visit, and the agents' life expectancy.

References

[1] H. Hamidi and K. Mohammadi, “Evaluation of Fault Tolerant Mobile Agents in Distributed Systems, “International Journal of Intelligent Information Technologies (IJIIT 5(1)), pp.43-60, Janauary-March 2009.

[2] H. Hamidi and A. Vafaei, "Evaluation of Security and Fault-Tolerance in Mobile Agents," Proc.Of the 5th IEEE Conf. on Wireless & Optical Communications Networks (WOCN2008), May 5, 6 and 7, 2008.

[3] H. Hamidi and K. Mohammadi, "Modeling and Evaluation of Fault Tolerant Mobile Agents in Distributed Systems," Proc. Of the 2th IEEE Conf. on Wireless & Optical Communications Networks (WOCN2005), pp. 91-95, March 2005.

[4] S. Pleisch and A. Schiper, "Modeling Fault-Tolerant Mobile Agent Execution as a Sequence of Agree Problems," Proc. of the 19th IEEE Symp. on Reliable Distributed Systems, pp. 11-20,2000.

[5] S. Pleisch and A. Schiper, "FATOMAS - A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach," Proc. 2001 Int'l Conf on Dependable Systems and networks, pp.215-224, 2001.

[6] M. Strasser and K. Rothermel, "System Mechanism for Partial Rollback of Mobile Agent Execution," Proc. 20th In!'l Conf on Distributed Computing Systems, 2000.

[7] T. Park, I. Byun, H. Kim and H.Y. Yeom, "The Performance of Checkpointing and Replication Schemes for Fault Tolerant Mobile Agent Systemss ," Proc. 21th IEEE Symp. On Reliable Distributed Systems, 2002.

[8] M. Izatt, P. Chan, and T. Brecht. Ajents: Towards an Environment for Parallel, Distributed and Mobile Java Applications. In Proc. ACM 1999 Conference on Java Grande, pages 15-24, June 1999.

[9] A. S. Tanenbaum, "Distributed Operating Systems", prentice Hall, Inc, 1995.

[10] H.W. Chan, K.M. Wong, R. Lyu “Design, Implementation, and Experimentation on Mobile Agent Security for Electronic Commerce Application,” Distributed systems, s. Mullender,ed., second ed., pp. 199-216, Reading, Mass.: Addison-wesley, 1993.

[11] X. Defago, A. schiper,and N. sergent, “semi-passive Replication,”proc. 17th IEEE symp. Reliable Distributed system (SRDS ‘98), pp. 43-50, oct. 1998.

[12] M.J. Fischer, N.A. Lynch and M.S. paterson, “Impossibility of Distributed consensus with one Faulty process,” Proc. second ACM SIGACT-SIGMOD symp. Principles of Database system, pp. 17-24, Mar.1983.

[13] M. S. Greenberg, J. C. Byington, and D. G. Harper. “Mobile Agents and Security”. In Volume 367, IEEE Communications Magazine. IEEE Press, July 1998.

[14] C. F. Tschudin. “Mobile Agent Security”. In M. Klusch, Intelligent Information Agents. Forthcoming LNCS. http://www.docs.uu.se/~tschudin/pub/cft-1999-iia.ps.gz, 1999.

[15] C. Ghezzi, G.Vigna. “Mobile Code Paradigms and Technologies: A Case Study”. In Kurt Rothermet, Radu Popescu-Zeletin, editors, Mobile Agents, First International Workshop, MA’97, Berlin, Germany, April 1997, Proceedings, LNCS 1219, p. 39-49. Springer, 1997.

[16] A. Fuggetta, G. P.Picco, & G.Vigna, Understanding code mobility. IEEE Transactions on Software Engineering, 24(5). pp. 342–361, 1998.

[17] W. Stallings. Cryptography and Network Security, Principles and Practice. Prentice Hall, 2nd edition, 1999.

[18] T. Sander and C. F. Tschudin. “Protecting Mobile Agents against Malicious Hosts”. In Giovanni Vigna, editor, Mobile Agents and Security, LNCS 1419, p. 44-60. Springer, 1998.

[19] F. Hohl. “Time Limited Blackbox Security: Protecting Mobile Agents from Mali cious Hosts”. In Giovanni Vigna, editor, Mobile Agents and Security, LNCS 1419, p. 92-113. Springer, 1998.

[20] M.K. Aguilera, w. chen, and s. Toueg, “Failure Detection and consensus in the crash-Recovery Model,” Distributed computing,vol. 13,no. 2,pp. 99-125,2000.

[21] R. A. Sahner, K. S. Trivedi, and A. Puliafito. Performance and Reliablity Analysis of Computer Systems. Kluwer Academic Publishers, Boston, 1996.

[22] S. Pleisch and A. Schiper, “Fault-Tolerant Mobile Agent Execution," IEEE TRANSACTIONS ON COMPUTERS, VOL. 52, NO .2, Feb 2003.

[23] M. Strasser and K. Rothermel, "System Mechanism for Partial Rollback of Mobile Agent Execution," Proc. 20th In!'l Conf on Distributed Computing Systems, 2000.

[24] H. Hamidi, “Modeling Fault Tolerant and Secure Mobile Agent Execution in Distributed Systems, "International Journal of Intelligent Information Technologies (IJIIT 2(1)), pp.21-36, 2006.

[25] H. Hamidi and A.Vafaei, “Evaluation and Check pointing of Fault Tolerant Mobile Agents Execution in Distributed Systems,” Journal of Networks (Academy Publisher), VOL. 5, NO. 7. July 2010.

[26] A. Vafaei, H. Hamidi., S.A. Monadjemi. “A Framework for Fault Tolerance Techniques in the Analysis and Evaluation of Computing Systems” International Journal of Innovative Computing, Information and Control (IJICIC), Vol.8, No.7, July 2012.

[27] S. Bimonte, L. Sautot, L. Journaux, & B. Faivre, Multidimensional Model Design using Data Mining: A Rapid Prototyping Methodology. International Journal of Data Warehousing and Mining (IJDWM), 13(1), pp.1-35. 2017.

[28] D. Chevers, A. Mills, E. Duggan, S. Moore. “An Evaluation of Software Development Practices among Small Firms in Developing Countries: A Test of a Simplified Software Process Improvement Model.” Journal of Global Information Management, 24(3), pp. 45-70. 2016.

[29] C. Esposito, & M. Ficco, “Recent Developments on Security and Reliability in Large-Scale Data Processing with MapReduce.” International Journal of Data Warehousing and Mining (IJDWM), 12(1), pp. 49-68. 2016.

[30] F. Gharagozlou, , G. A. Mazloumi, , A. Nahvi, Nasrabadi, A. M., Foroushani, A. R., Kheradmand, A.A, Ashouri, M., Samavati, M, Detecting Driver Mental Fatigue Based on EEG Alpha Power Changes during Simulated Driving, Iranian Journal of Public Health 2015.

[31] H. Hamidi, “A Combined Fuzzy Method for Evaluating Criteria in Enterprise Resource Planning Implementation.” International Journal of Intelligent Information Technologies (IJIIT), 12(2), pp.25-52. 2016.

[32] Hamidi, H. “A Model for Impact of Organizational Project Benefits Management and its Impact on End User”, JOEUC, Volume 29, Issue 1, pp. 50-64, 2017.

[33] R. D. Johnson, Y.Li, & J. H. Dulebohn, “Unsuccessful Performance and Future Computer Self-Efficacy Estimations: Attributions and Generalization to Other Software Applications.” Journal of Organizational and End User Computing, 28(1), Pp.1-14. 2016.

[34] A. S. Kakar, "A User-Centric Typology of Information System Requirements." JOEUC, 28(1), pp. 32-55. 2016.
[35] S. Kumar, "Performance Evaluation of Novel AMDF-Based Pitch Detection Scheme," ETRI Journal, vol. 38, no. 3, pp. 425-434, 2016.
[36] Y. Liu, C.Tan, & J. Sutanto, "Selective Attention to Commercial Information Displays in Globally Available Mobile Application." Journal of Global Information Management, 24(2), pp.18-38. 2016.
[37] S.A. Monadjemi, A.Vafaei, H.Hamidi, "Analysis and Evaluation of a New Algorithm Based Fault Tolerance for Computing Systems." International Journal of Grid and High Performance Computing (IJGHPC), 4(1), pp. 37-51, 2012.
[38] S.A. Monadjemi, A.Vafaei, H.Hamidi. "ANALYSIS AND DESIGN OF AN ABFT AND PARITY-CHECKING TECHNIQUE IN HIGH PERFORMANCE COMPUTING SYSTEMS" Journal of Circuits, Systems, and Computers (JCSC), JCSC Volume 21 Number 3. 2012.
[39] A. Safdar, K. DoHyeun, "Enhanced power control model based on hybrid prediction and preprocessing/post-processing." Journal of Intelligent & Fuzzy Systems, vol. 30, no. 6, pp. 3399-3410. 2016.
[40] B. Shadloo, A. Motevalian, V. Rahimi-movaghar, M. Amin-Esmaeili, V. Sharifi, A. Hajebi, R. Radgoodarzi, M. Hefazi, Rahimi-Movaghar, "Psychiatric Disorders Are Associated with an Increased Risk of Injuries: Data from the Iranian Mental Health Survey." Iranian Journal of Public Health 45(5): pp. 623-635. 2016.
[41] J. Wu, F. Ding, M. Xu, Z. Mo, & A. Jin, "Investigating the Determinants of Decision-Making on Adoption of Public Cloud Computing in E-government." JGIM, 24(3), pp.71-89. 2016.
[42] X. Ye, T. Sakurai, "Robust Similarity Measure for Spectral Clustering Based on Shared Neighbors," ETRI Journal, vol. 38, no. 3, pp. 540-550. 2016.

Hodjat Hamidi born 1978, in shazand Arak, Iran, He got his Ph.D in computer engineering. His main research interest areas are Information Technology, Fault-Tolerant systems and applications and reliable and secure distributed systems and e- commerce. Since 2013 he has been a faculty member at the IT group of K. N. Toosi University of Technology, Tehran Iran. Information Technology Engineering Group, Department of Industrial Engineering, K. N. Toosi University of Technology.

Maryam Parvini is a master student of Information Technology Engineering at K. N. Toosi University of Technology. Her research interests include Machine learning, Knowledge Discovery and Data Mining and Customer Relationship Management.