Dependable Mobile Agents

ADEL CHERIF and TAKUYA KATAYAMA
School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Asahidai Tatsunokuchi 923-1211 Nomi-Gun
JAPAN

Abstract: - The mobile agent programming paradigm offers several advantages for the development of distributed systems and applications. Mobile agent dependability is a key requirement for the acceptance and future development and deployment of mobile agents. In this paper, we propose a replication scheme for mobile agents based on the group concept and associated group communication services. Group communication has been proposed to support fault tolerance in traditional distributed systems and proved to be very effective in the design and implementation of fault tolerant and dependable systems. We argue that it is also attractive for the design and implementation of dependable mobile agents.

Key-Words: - Dependability, Exactly-once Execution property, Mobile Agents, Group communication.

1 Introduction

The ubiquitous use of the network and the Internet and the increasing interest and adoption of wireless networking as well as the increasing popularity of mobile computing devices such as laptops, PDAs and Palm devices is leading to the establishment of a new computing environment that is highly dynamic, heterogeneous, flexible and highly mobile. The mobile agent paradigm is a versatile and robust programming paradigm that offers several advantages for the design and development of distributed systems and applications deployed on such distributed computing environment.

An agent is a software program that is executing on behalf of its owner. The owner can be either a user or a software program. A mobile agent is an agent that can migrate autonomously from node to node in the distributed system to perform computation on behalf of its owner. In order for the mobile agent to execute and run at a node, the visited node must support an agent execution environment also called agent system. In the remainder of this paper, we will refer to such an execution environment as an agency. An agency offers access to services and resources that the mobile agent needs in order to execute. Several Mobile agent systems have already been proposed in the literature including Aglets from IBM [1] and Concordia from Mitsubishi [2]. Some of the developed systems are commercially distributed while others are freely distributed on the Internet.

The very nature of the modern distributed computing environment makes it highly prone to failures and recoveries. Failures can lead to the loss of mobile agents and blocking since no progress can be achieved until the failed mobile agent recovers. The crash of the mobile agent itself, or the agency, or the node on which the mobile agent was executing can result in the loss of the mobile agent. Recovery, in the other hand, can lead to mobile agent duplication, that is, the creation and execution of two or more instances of the same mobile agent. In such a case, tasks assigned to the original instance of the mobile agent might be executed more than once. For some type of applications, such as for electronic commerce applications, this clearly violates the application requirements such as the exactly-once execution property [8].

Ensuring mobile agent dependability despite failures and recoveries is of primary importance for the acceptance and future development and deployment of mobile agent based applications. To provide dependable mobile agent, one must ensure that despite failures and recoveries, mobile agents will not be lost and that, when required, the mobile agent will be executed at most once, that is, it verifies the exactly-once execution property.

In the distributed computing environment, it is not always possible for the owner to distinguish between the three following cases: The mobile agent has crashed or is just executing slowly or is temporarily unreachable due to a communication link failure or network partition. Thus, the mobile agent owner cannot reach a decision on the mobile agent status. This problem is also inherent to traditional distributed computing. Strategies and techniques that proved to be efficient in traditional distributed computing in ensuring system dependability should also be considered for the implementation of dependable mobile agents.

In this paper we present a replication scheme based on the process group approach [3] and on group communication [4] for implementing dependable mobile agents. The proposed replication scheme prevents blocking and ensures that mobile agents are not lost as a result of failures. It will also ensure that, when required, mobile agents will not violate the exactly-once execution property.

The remainder of this paper is organized as follows: Section 2 introduces the related work. The system, the failure, and the mobile agent execution models are described in section 3. The proposed replication scheme is introduced in Section 4. Finally, section 5 offers some concluding remarks.

2 Related work

An increasing number of research work is being conducted on developing techniques and approaches to ensure mobile agents dependability. Some of the approaches proposed in the literature are based on checkpointing and recovery [5, 6] while others are based on replication [7,8,9,10].

In the checkpointing and recovery techniques, a copy of the mobile agent is saved in persistent or stable storage. In case of crash of the agent, the saved copy is retrieved from storage and the mobile agent is restarted from the last saved checkpoint. If the agency or the node fails leading to the crash of the mobile agent, the mobile agent is also retrieved from persistent storage and restarted after the node and/or the agency recovers. Other techniques use a more elaborate approach where before migrating to a new node N_{c+1} a copy of the mobile agent is saved at the current node N_c. This copy is on stand-by mode and acts as a back-up to recover the agent if it fails during migration or if it fails to restart at the new node. The migration process can be processed as a transaction, that is, it is executed atomically verifying the all-or-nothing property. In case of failure of the mobile agent, a copy saved at one of the previous nodes is activated and restart execution. Some approaches as in [5] maintain these back-up copies until the agent completes its tasks and then garbage collect them. Other approaches as in [6], delete the copy after the next migration, that is, when the mobile agent successfully migrates form node N_{c+1} to a new node N_{c+2}.The copy saved at node N_c is garbage collected.

Some of the proposed checkpointing and recovery techniques do not prevent blocking and most do not prevent mobile agent duplication and thus can lead to the violation of the exactly-once execution property.

Several replication schemes [7,8,9,10] have been proposed in the literature to implement dependable mobile agents.

In [7], the author uses replication and voting to mask the effects of failures. The author focuses mainly on security and on protecting the mobile agents against malicious hosts. This work is based on the state machine approach [11]. The performance cost of such approach is very high.

In [8], the authors present a fault tolerant protocol that ensure the exactly-once execution of agents. This work is based on the primary/back-up approach [12]. In the proposed model, agents are replicated and one replica is called the worker while other replicas are called observers. In this approach, an agent executes a single atomic transaction at each agency it visits. A voting protocol integrated with the two-phase commit protocol is used by the worker to commit the transaction. In this approach, the failure of a single node can lead to blocking.

In [9], the authors present a variation of the protocol introduced in [8] to support the execution of more than a single transaction at the visited agency. The authors assume a fail-silent failure model and they also base their approach on the primary/back-up approach. The authors assume reliable communication channels and use a three-phase commit protocol combined with the voting protocol in order to alleviate the possibility of blocking due to a failure of a single node. The three-phase commit protocol requires extra rounds of communication between the worker and the observers and the protocol can block if the communication channels are unreliable.

In [10], an agent-dependent approach is proposed. In the agent-dependent approach, and unlike agency-dependent approaches where the fault tolerance protocols are supported at the agency and are integrated in the agency itself, the fault-tolerance mechanisms are implemented at the mobile agents. In the proposed approach, the mobile agent is replicated on a number of nodes and the fault tolerant execution of the mobile agent leads to a sequence of agreement problems between the agent replicas. An instance of the DIV Consensus algorithm is then executed at each execution stage to solve the agreement problem. The agent-dependent approach increases the overhead associated with the agent migration since the code and data used for the FT protocols are part of the agent. Furthermore, solving the consensus problem at each execution stage has a high performance cost. Finally, such agent-dependent approaches do not scale nicely.

3 System Model

We consider a distributed asynchronous system consisting of a number of nodes connected by a communication network as shown in Fig.1. In an asynchronous system no assumptions can be made on the communication delays or processing speeds. In the considered model, processes communicate solely by means of message passing.

3.1 Failure Model

We assume that the communication network may partition. A network partition occurs when some nodes in the system can communicate with each other but cannot communicate with other nodes in the system. We assume that messages and agents are not corrupted during their transmission on the communication network.

The failure model supported by the proposed protocol is the crash failure model for mobile agents, agencies and nodes. In the crash failure model, it is assumed that mobile agents and/or agencies and/or nodes behave correctly or fail by simply halting without producing incorrect results.

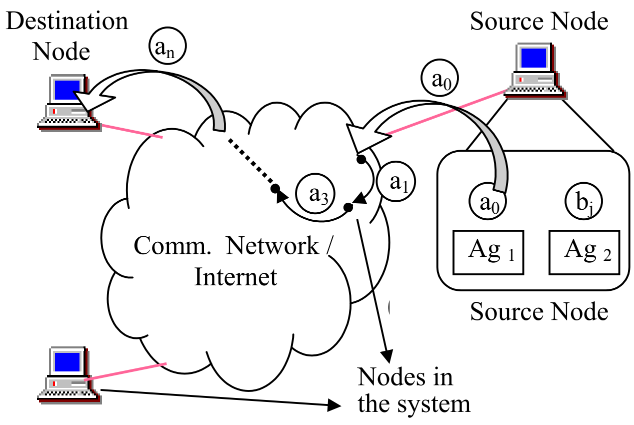

Fig.1, System and Agent Execution Models

3.2 The Mobile Agent Execution Model

Each node in the system hosts one or more agencies. Different type of agencies (Ag_i) can coexist at the same node (e.g., Aglet and Concordia agencies). A mobile agent is initially created at a node called the source node. It then migrates from node to node in the system to perform its assigned tasks until it reaches its final node called the destination node as shown in Fig.1. The destination node can be the same as the source node. Each node visited by the mobile agent must host an agency of the same type than the mobile agent. That is, an Aglet mobile agent can only be executed by the agency of type Aglet.

A mobile agent (a) is assigned a number of tasks n that it has to perform. In the remainder of this paper, we will denote a mobile agent (a) that performed a number of tasks i where $(0 \leq i \leq n)$ as a_i. In the considered model a mobile agent can execute any number of tasks at each node it visits, then it eventually migrates to another node in the system. The agency executes the mobile agent resulting in a new internal state for the mobile agent. Some of the executed tasks might change the state at the visited node. For example, a mobile agent that simply reads some information from a database server will not change the state of the server. However, a mobile agent that performs a money withdraw operation from a bank server that is resident at the visited node clearly changes the state of the server. Such tasks must be executed exactly once by the mobile agent, we call such tasks critical tasks.

4 The Proposed Replication Scheme

In order to implement dependable mobile agents, there is a need to add some form of redundancy into the system. Replication has proved to be one of the most powerful techniques to provide fault tolerance and system dependability. The Process group concept and group communication services also proved to be very useful in designing and implementing dependable distributed systems. The proposed replication scheme is based on passive replication also known as the primary/Backup approach and on group communication services.

4.1 Replicated Mobile Agent Execution Model

When first created, the mobile agent is replicated and a number of identical copies of the mobile agent are then created. Each instance of the replicated mobile agent is called a replica. A new group with a system-wide unique identifier is then created and all replicas are joined into the group. A ranking is assigned to each replica in the group with the original mobile agent having the lowest ranking. A mobile agent a with rank j is denoted by a^j. A mobile agent a with rank j which state reflects the execution of i tasks is denoted by a_i^j. The mobile agent with the lowest ranking in a group is called the primary agent. Other replicas are called back-up agents. After all replicas are joined into the group, they are dispatched to different nodes in the system as shown in Fig. 2.

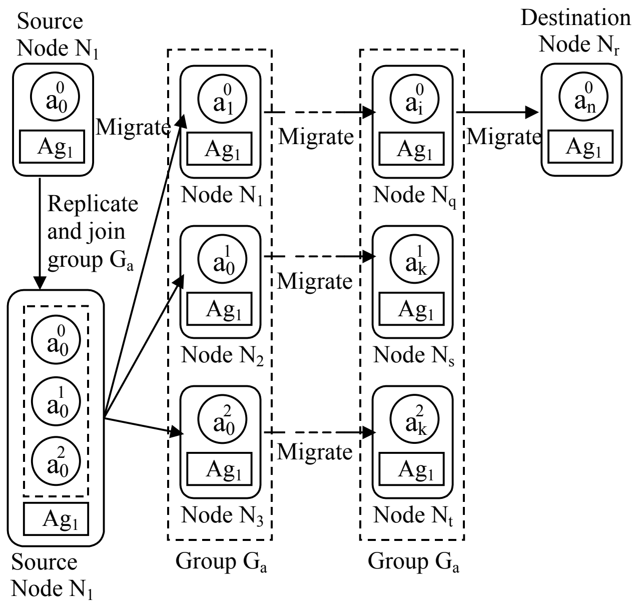

Fig.2, Replicated Mobile Agent Execution and Migration Model

There is only one primary agent at any time in any group. The primary agent is the only active mobile agent in its group. That is, it is the only mobile agent that executes the tasks assigned by the owner. Other back-up agents are in stand-by mode. All members of a particular group are located at different nodes. When the primary agent migrates from a node to another in the system, back-up agents may also migrate to different nodes in the system in order for example to minimize communication costs and thus improve system performance.

4.2 The Replication Scheme

In the proposed replication scheme, A membership service manages all groups in the system. The membership service continuously monitors group members. All members of a group have the same view on their group membership provided by the membership service.

In case of failure of a group member, the failing member is removed from the group and a new group view is generated. If the failing member is a back-up agent, depending on the number of replicas in the group a new back-up agent might be created in order to maintain the required level of replication. To create a new back-up agent, one of the back-up agents is cloned and the new back-up agent is dispatched to another node in the system. If the failing agent is the primary agent, a new group view excluding the failing member is also generated and the agent with the lowest ranking becomes the new primary agent and starts execution.

In case of network partition, a group may be divided into a number of subgroups. We assume that there always exists a subgroup that includes at least half of the members of the partitioned group. The replication protocol ensures progress only in one connected group, called the primary group as shown in Fig.3. The primary group consists of the group with a number of agents that is at least equal to half the number of agents included in the previous primary group. If a group is partitioned into two groups with an equal number of agents, then the group including the primary agent from the previous primary group is considered as the new primary group. Agents in other minority groups are simply blocked and garbage collected. If the primary agent is a member of a minority group, it is also blocked and one of the back-up agents in the new primary group will become the new primary agent.

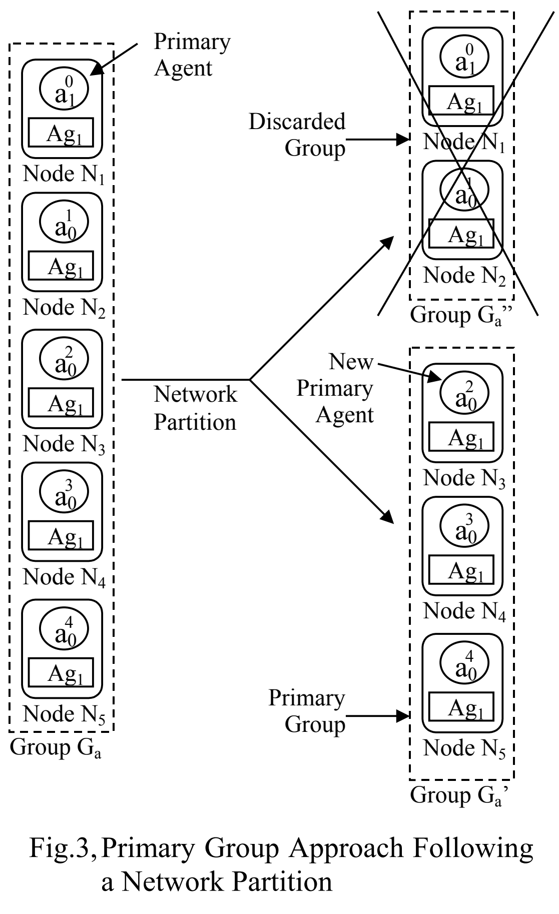

Fig.3, Primary Group Approach Following a Network Partition

The state of the back-up agents is updated periodically by the primary agent or when the primary agent needs to execute a critical task. The execution of a critical task by the primary agent modifies the state of the node on which the primary agent is executing. This task is executed as a transaction and the protocol must guarantee that this execution will occur at most once despite failures and recoveries. To do so, all group members participate in a two-phase commit protocol to commit the transaction. The protocol avoids blocking by using the membership service to continuously monitor the agent group membership. In case of failure of one of the members, the transaction is aborted and the commit protocol is restarted again possibly after a group reconfiguration is performed and a new group membership view is generated by the membership service.

Once the transaction is committed the state of all group members will reflect the execution of the critical task. As a result, in case of failure of the primary agent, the new primary agent will not execute the same task again. For idem-potent tasks, that is, tasks that can be executed more than once, there is no need to execute the commit protocol. The primary agent periodically updates the state of the back-ups.

4.3 System Support for Replication

To support the proposed replication scheme, a runtime system consisting of a replication manager and a group communication manager is resident at each node in the system. The replication manager supports the mobile agent group paradigm and provides common group operations such as create, join and leave group. It is responsible for creating back-up agents to maintain the required level of replication and to manage these agents. It also implements a two-phase commit protocol.

The replication manager uses the services provided by the group communication manager to perform its operations. The group communication manager offers a membership service and an atomic multicast service. The membership service maintains information on the groups existing in the system and their members. It continuously monitors the members of the mobile agent group and notifies the replication manager of any failures. The group communication service provides a communication primitive that guarantees the delivery of messages in the same order at all group members.

5 Conclusion

We proposed a replication scheme based on group communication for the design and implementation of dependable mobile agents. The proposed replication scheme prevents the blocking of mobile agents following the crash of the mobile agent itself or the agency or the node on which it is executing. It also ensures the exactly once execution property that is required for some type of applications including electronic commerce.

References:

[1] Programming and deploying Java Mobile Agents with Aglets. Danny Lange and Mitsuru Oshima, Addison Wesley, 1998.

[2] D.Wong, N.Paciorek, T.Walsh, J.DiCelie, M.Young, B.Peet. Concordia: An Infrastructure for Collaborating Mobile Agents. *LNCS*, Volume 1219, Springer, 1997.

[3] K.P.Birman. The Process Group Approach to Reliable Distributed Computing. *In Communications of the ACM*, No.12, Vol.36, pp:37-53, 1993.

[4] F.Cristian. Synchronous and Asynchronous Group Communication. *In Communications of the ACM*, No.4, Vol.39, pp:88-97, 1996.

[5] L.Silva, V.Batista, and J.G.Silva. Fault-Tolerant Execution of Mobile Agents. In Proc. of the International Conference on Dependable Systems and Networks, pp:135-143, 2000.

[6] D. Johansen, R.van Renessee, and F. Schneider. Operating System Support for Mobile Agents. In Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems.

[7] F.B. Schneider. Towards Fault-tolerant and Secure Agentry. *LNCS,* Volume 1320, Springer, 1997.

[8] M.Straber, K.Rothermel, C.Maihofer. Providing Reliable Agents for Electronic Commerce. *LNCS*, Volume 1402, Springer, 1998.

[9] F.A.Silva and R.Popescu-Zeletin. An Approach for Providing Mobile Agent Fault Tolerance. *LNCS,* Volume 1477, Springer, 1998.

[10] S.Pleisch and A. Schiper. FATOMAS: A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach. In Proc. of the International Conference on Dependable Systems and Networks, 2001.

[11] F.B.Schneider. Replication Management Using the State-Machine Approach. *Distributed Systems*, Sape Mullender, editor, pp:169-197, ACM Press, 1993.

[12] N.Budhiraja, K.Marzullo, F.B.Schneider, and S.Toueg. The Primary-Backup approach. *Distributed Systems*, Sape Mullender, editor, pp:199-216, ACM Press, 1993.