Chapter 9
Evaluation of Fault Tolerant
Mobile Agents in Distributed
Systems

Hojatollah Hamidi
University of Isfahan, Iran-Isfahan

Abbas Vafaei
University of Isfahan, Iran-Isfahan

ABSTRACT

The reliable execution of a mobile agent is a very important design issue to build a mobile agent system and many fault-tolerant schemes have been proposed. Hence, in this article, we present an evaluation of the performance of the fault-tolerant schemes for the mobile agent environment. Our evaluation focuses on the checkpointing schemes and deals with the cooperating agents. We derive the Fault-Tolerant approach for Mobile Agents (FANTOMAS) design which offers a user transparent fault tolerance that can be activated on request, according to the needs of the task. We also discuss how a transactional agent with different types of commitment constraints can commit. Furthermore, this article proposes a solution for effective agent deployment using dynamic agent domains.

INTRODUCTION

The client/server computing paradigm is today’s most prominent paradigm in distributed computing. In this computing paradigm, the server is defined as a computational entity that provides some services. The client requests the execution of these services by interacting with the server. Having executed the service, the server delivers the result back to the client. The server therefore provides the knowledge of how to handle the request as well as the required resources. The computing paradigm of mobile code generalizes this concept by performing changes along two orthogonal axes:
Evaluation of Fault Tolerant Mobile Agents in Distributed Systems

1. Where is the know-how of the service located?
2. Who provides the computational resources?

Depending on the choices made on the client and server sides, the following variants of mobile code computing paradigms, illustrated in Table 1, can be identified (Fuggetta, Picco, & Vigna, 1998):

In the Remote Evaluation (REV) paradigm, component A sends instructions specifying how to perform a service to component B (represented by code in Table 1). 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 (Coward, 2001).

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 (represented by code in Table 1). As soon as A has the necessary know-how (i.e., has downloaded the code from B), it can start executing. Java applets fall under this variant of the mobile code paradigm.

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 Machine. 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 machine. 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 machines, i.e., can make multiple hops.

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

| Table 1. Different variants of the mobile code computing paradigm (Fuggetta, Picco, & Vigna, 1998). Code or computational entity transported between machines are indicated by italics. Component A accesses the services provided by component B. |
|---|---|
| before the invocation | after the invocation |
| machine 1 | machine 2 | machine 1 | machine 2 |
| Client/Server | A | code,resource,B | A | code,resource,B |
| Remote Evaluation | code,A | resource,B | A | code,resource,B |
| Code on Demand | resource,A | code,B | code,resource,A | B |
| Mobile Agent | code,A | resource,B | code,resource,A | B |
Related Content

Application of Text Mining Methodologies to Health Insurance Schedules
Ah Chung Tsoi, Phuong Kim To and Markus Hagenbuchner (2006). Advances in Applied Artificial Intelligence (pp. 29-51).
www.igi-global.com/chapter/application-text-mining-methodologies-health/4672?camid=4v1a

An Intelligent Operator for Genetic Fuzzy Rule Based System
www.igi-global.com/article/intelligent-operator-genetic-fuzzy-rule/58054?camid=4v1a

Using a Business Ontology for Structuring Artefacts: Example - Northern Health
www.igi-global.com/article/using-a-business-ontology-for-structuring-artefacts/142900?camid=4v1a

Auto-Associative Neural Networks to Improve the Accuracy of Estimation Models
www.igi-global.com/chapter/auto-associative-neural-networks-improve/36442?camid=4v1a