A Framework for Mobile Agent Platform performance Evaluation

Rocco Aversa, Beniamino Di Martino, Massimiliano Rak and Salvatore Venticinque
Dipartimento di Ingegneria
dell’Informazione,
Seconda Università di Napoli
{massimiliano.rak, salvatore.venticinque}@unina2.it

Abstract

Mobile agents programming paradigm is an emerging approach for distributed computing, extremely suitable for mobile systems because of its adaptability to exploit the available resources. Optimization of mobile agents applications and system configuration are relevant above all when we deal handheld devices with limited capabilities. Classical approaches are hard to apply because new kinds of interaction facilities provided by agent platforms, such as cloning and migration, represent an additional software layer that affects the system performances. In this context identification and estimation of performance indexes are necessary to evaluate and foresee system dependability. In fact performance evaluation is exploited to address different issues according to which, different measurements are required. In practice, there is a need for tools and techniques for evaluation of the performances of the adopted mobile agent platforms. Related work proposes a set of performance indexes, slightly different one from the other, and measured using different approaches. We present here a framework that aims at supporting the development of ad-hoc solutions based on measurement agents. Framework architecture, prototype implementation and case studies and preliminary performance figures are presented in the following.

1 Introduction

Mobile agents programming paradigm is an emerging approach, in which computation is delegated to software entities able to migrate themselves together with their code and state. Agent platforms, are emerging as good solutions in many fields, like GRID [15, 18, 9] or SOA (Service Oriented Architecture)[11, 10]. This approach is extremely suitable for mobile systems due to the its adaptability to the available resources; dedicated platforms for wireless mobile devices exists, like JADE-LEAP[2, 3]. However when agents mobility has been exploited a lot of properties of the dependability of the final application are affected. Security is critical as both agents and hosting nodes must be protected during migration. Mobile agent approach helps in building reliable applications, but ad hoc solution needs to be defined to face problems like the reliability of destination hosts where agents are being dispatched, which is often unknown. Here we focus on performance that becomes a difficult property to be evaluated because mobile devices are provided with limited resources and it is hard to apply classical techniques for system optimization (such as ad-hoc tuning, performance engineered software development, ...). Continuous changes of the execution contexts, i.e. due to the agents migration from a node to another make difficult to predict dynamics of workload distribution and system utilization. New kinds of interaction facilities such as cloning and migration are largely exploited. They are provided by agent platforms, which represent an additional software layer that affects the system performances. Moreover there is a need for defining how the underlying hardware architecture affects the platform behavior. So even if a good approach may help in producing good applications in terms of performances, in practice, there is a need for tools and techniques for evaluation of the performances of the adopted mobile agent platforms. In this paper we present a tool that supports the design and the benchmark of mobile agents platforms. The idea is to provide a framework for development of measurement agents. The framework aims at simple and flexible development of set of agents able to build up a testbed and a set of agents for measurement purposes. The framework may be of help in development of agents dedicated to performance measurement in fault conditions.

The reminder of the paper is organized as follows: next section proposes an analysis of the research context, de-
2 Research Context

2.1 Mobile Agents

A mobile agent is a Software Agent with an added feature: the capability to migrate across the network together with its own code and execution state. This paradigm allows both a pull and a push execution model [20]. In fact, the user can choose to download an agent or to move it to another host. Mobility can provide many advantages when we aim at developing distributed applications. System reconfiguration by agent migration can help to optimize the execution time by reducing network traffic and interactions with remote systems. Furthermore, statefull migration allows to redistribute dynamically the agents for load balancing purposes. Several different criteria can guide agent distribution, such as moving the execution near to the data, exploiting new idle nodes, or allocating agents on the nodes in such a way that communications are optimized. Two forms of mobility characterize the migration of agents:

- **Strong mobility** is the ability of a Mobile Code System (called strong MCS) to allow migration of both code and full execution state of an agent (this concept is very close to process migration).

- **Weak mobility** is the ability of an MCS (called weak MCS) to allow code transfer across different Computing Environments (CEs) together with the intermediate data that are relevant to resume the status of the application. Neither program counter and other information about the process status are dispatched.

In the following, we will consider only weak MCS, which is the only adopted on mobile devices, at the authors knowledge. It is relevant as performance on different architectures is affected by the virtual machine used to interpreter the intermediate agent code (Java or similar languages). In order to support agent execution, each node in a computing network must run a server application that provides the environment where agents can be hosted (the agent platform). The environment, called agent place or agent container, provides also the facilities necessary to interact with other agents or with the environment itself. Typical forms of interaction are communication among agents, or exploitation of mechanisms such as cloning, creation, disposal and migration. Currently there are two standards that define the architecture of a mobile agent platform. The MASIF [13] specifications deal with the interoperability among different implementations of a mobile agent platform. MASIF defines by a CORBA-like syntax some standard interfaces which should be implemented to provide uniform access to services such as agent migration, cloning, creation. It also defines how agents must be identified inside a platform (name, origin, ...). The FIPA standard [8], on the other side, focuses on the design of a standard architecture of an agent system aiming at granting the interaction among agents belonging to the same platform or to different ones. It describes the software architecture of an agent platform, the basic services which should be provided, the life cycle of an agent and the agent communication language. Many different implementations of the mobile agent programming paradigm are available. Due to previous experiences and for the facilities it offers, we chose to adopt the JADE Platform, a FIPA compliant agent platform developed by TILAB [1].

2.2 Performance Evaluation

Performance evaluation analysis of mobile agents platforms focuses on two main problems: comparison of different agents solution or optimization of a target platform. In both cases exists the problem of defining a good set of performance metrics, strictly related to the mobile agent paradigm. In terms of comparison between different platforms the most common approach is the adoption of benchmarking programs, i.e. common or ad-hoc developed applications measured in terms of response time, as an example [12] compares Aglets and Tacoma platform upon a distributed search algorithm. As an alternative [17] compares a large set of platforms using a simple roaming agents. This kind of analysis are of little use when the benchmark is not standard. At the state of the art no standard benchmark exists. Dikaiakos et al.([5, 6]) proposes an alternative approach, building a structured (hierarchical) benchmarking approach which aims at stressing the target platforms at different level, from fine grained (micro-benchmarks) to large grain (application level). In [16] they apply the proposed approach in order to evaluate the adoption of agents for database access. Gavalas et al. [7] focus on the problem of platform optimization, they try to reduce the agent migration time, optimizing the agent serialization and the network communications. Performances of the JADE platform are proposed in [19], which focuses on the message exchange between agents, both inter- and intra- platforms. A more complete study in [4] proposes a large set of tests, Spamming test, DataBase access, agent creation and migration. A systematic analysis applying Design of Experiment technique is proposed in [14], which tries to point out the role of Operating System and JRE on the platform performances. It focuses on Secondary containers startup time, on agent
migration and AMS interrogation (measured as response time). All the cited works propose different set of performance indexes, slightly different one from the other, and measured using different approaches. Performance evaluation can address different issues according which different measurements are required. The tool we propose aims at supporting the development of ad-hoc solutions.

3 The Proposed Framework

The framework architecture we designed has been conceived to develop and use benchmarks for FIPA compliant agent systems. The target is to provide a software toolset for test developing and performance analysis of different components and functionalities of a Mobile Agent Platform. The proposed solution has been provided for the JADE platform. Figure 1 shows the framework architecture; it is composed of a software engine called core and of additional plug-ins specialized to execute specific benchmarks. Testbed configuration data represent the input of each analysis. A graphical user interface can be exploited in order to manage the experiments and to evaluate the results. At the state of the art the core of the framework offers facilities for platform configuration, i.e. tools which help the user to configure the architecture under test, the test configuration, i.e. tools which help the user to configure the specific benchmark he is interested on, and results visualization and management. Platform configuration means that the framework let the user to describe how the jade platform is deployed upon the underlying executing environment: JADE platforms is organized in Containers, which are the software agents which are able to host the mobile agents, the Main Container, which is the first started by the user, can be queried in interactive way from user and is able to start up new containers on other computing nodes. The framework offers features to build up a predefined execution environment, i.e. starts up a set of containers on given set of working nodes, in which the tests will be executed. All the configurations are made in XML, in the following we will give some examples. Test Configuration means that all the tests are parameterized, and it is possible to customize their behavior. The tests are developed into plug-ins, and the configurations details and parameters are defined into the plug-in development. All the configurations are always done in XML. In order to help performance data management and results visualization, the framework offers a set of agents which store the data and are able to perform simple statistical evaluations (calculate mean, variance, ...)

3.1 Framework Architecture

As previously introduced the framework is organized with a central core, which offers a set of common features and with a collection of plug-ins which offers specific tests. The framework is agent-oriented, so its software components are mostly jade agents, even if some of them are configuration files and other kind of tools. In the following we will use the term components to indicate any kind of software (configuration file, external tool, classes, graphical interfaces, ...) which is adopted and offered by the framework, while we will explicitly say agent, when the component is a jade mobile agent. The core is composed of the following components:

**Interface:** It handles the configuration. The configuration is described by an XML file It is able to provide a snapshot of the experiment during the test.

**Coordination:** It is implemented by an agent able to interact with the other parts of the system. It is initialized by the interface with the test configuration. When the interface asks for a snapshot of the system it collects the required parameters (number of nodes, amount of free memory, active threads, ) interacting with the other components

**Configuration file:** It describes the architecture to be tested and the experiments to be executed. Each experiment refers to a specific plug-in and provide the parameters for its initialization. It represents the interface between core and plug-in components.

The framework agents are classified in two main categories: management and testing agents. The management ones provide facilities for interaction with the user, management of the configuration, management of the experiments and results collection. The testing components perform a specific benchmark and produce the results. The core offers a fixed set of management agents (for platform management and user interface), while the plug-ins offers both management agents (for testing configuration and specialized data collection) and testing agents. Figure 2 summarize the subdivision of components between core and plug-ins. The frame-
work not only offers a set of stable software components which help in setting up the testing environment and coordinate the plug-in specific agents, but offers a set of base objects and agents which are extended by the plug-ins in order to specialize the benchmarking procedure; so all the plug-ins contain the following components:

**Configurator:** it is an agent that manages the single benchmark. It creates the infrastructure to be tested interacting directly with the core.

**TestManager:** A benchmark can be configured to repeat multiple experiments. Each experiment is a test-set performed through a cycle composed of two different phases. The test configuration phase creates the components which will execute the tested if they are not already available (creation of containers, creation and initialization of agents). The execution phase during which the experiment needs to be started, to be supervised and to be closed by managing all the actors.

**DataCollector:** It collects the results of the experiments for each execution providing the output that will be read by the interface.

**TestExecution:** It performs the real experiment. It can organized developing more components which are managed by the TestManager and interact among each with the others to conclude the experiment. They are completely independent from the core.

The tests offered by plug-ins may be composed by platform and test configurations. Figure 3 shows the an example of framework configuration to perform multiple experiments. Note that each experiment is defined by a name and consists of plug-in specific configuration. In order to configure the experiment the user must list the agents which perform the experiment providing and their initial parameters if it is necessary. A parser processes the configuration files and creates the agents which belong to the first experiment providing each one with the defined arguments. In figure 4 we can see how the agents are created and how they interact in order to complete an experiment. The ConfigAgent can start to initialize the platform right after its creation. It also creates the agents which will execute the test. Then the ConfigAgent asks to the AMS (the agent management system in charge of managing information about agents and container of the platform) for the completion of the configuration. Hence the TestManager is notified in order to start the test. The TestManager send the sig-
4 Framework Application Examples

In order to show how to use the proposed framework, we discuss some interesting results we obtained. We evaluated the migration time and the communication time of agents in the Jade platform. We also show early results even if at the state of the art we have not investigated yet the reasons about the obtained performance figures. Each plug-in has been developed by extending some base classes provided by the framework. These classes provide methods to communicate with the core and with the others agents in order to activate the protocol described in the previous section. Due to lack of space we do not provide here a detailed description of the available APIs and the developed code, but we prefer to describe how the framework has been configured and used. All the tests proposed in this paper were applied in the PARSsec Research group laboratory at Engineering Faculty of the Second University of Naples. The testing environment we adopted is Orion, an IBM blade cluster composed of 7 nodes, each of them with two Intel Xeon, 1 Gb RAM, 72 GB hard disk, two (Giga) Ethernet. The system is managed using the Rocks Linux distribution.

4.1 Migration Plugin

Past experiences in performance evaluation of mobile agents platforms defined various performance indexes to evaluate agent migration capability (see section 2.2. Here we define $T_m(\text{path})$, where $\text{path}$ is an ordered set of nodes, as the migration time that an agent needs to complete the migrations across all the nodes which belong to the given path. We assume that the paths are always closed, i.e. the agent starts from a given container (usually the Main container) and come back to the same container. In this experiment $T_m$ depends on:

- $\text{agentSize}$: size of the agent;
- $\text{Ncont}$: number of container and covered path;
- $\text{Nnodes}$: number of hosts;
- $M(\text{Ncont, Nnodes})$: mapping of containers on different nodes.

$$T_m = T_m(\text{Ncont, Nnodes})$$

The configuration file for the developed the plug-in is showed in Figure 5. The Configurator will set up 3 container for the experiment. The TestManager will start a certain number of test. We can see the description of the first one that will be performed 30 times with an agent size of 1024 KB. We have just a TestAgent that will execute the test by migrating along the path defined in the current TestSet and collecting time measures. The data collector is in charge of receiving the data from the TestAgent and to provide the processed results in the output file provided as a parameter. Containers can be allocated on nodes according two different policies: round robin, max fixed. According the round robin allocation policy if $\text{Nnodes} = 2$, when $\text{Ncont} = 2$ we create a container on each node. When $\text{Ncont} = 3$ we create two container on the first node and one container on the second one. In the max fixed policy we define a maximum number of containers for each node. The containers are created on the first nodes till the maximum number is reached. After that the other available nodes are exploited in the same way. We observed that when the size is less than some hundreds KB the operative
system affect the measure. When the size overcome 10MB we notice some errors in the JVM execution. So in our experiments the size is varying from 1 to 10 MB. Experiments showed that having more than one container on a node does not affect in a significant way Tm index. Hence we will present only performance figures allocating only one container per node. Furthermore experiments with more than 4 nodes cause JVM error because of out of memory. Figure 6, 7 show how the migration capability of an agent is affected by the agent size and the platform distribution. In Figure 6 the migration time increases with the agent size, but a different behavior can be observed when the size is 4 MB and 5 MB. Moreover different behavior the platform has different behaviors for a number of nodes less or greater than two, as shown with the results for agents between 1 MB up to 5 MB. We are still finding the reasons going to analyze the Jade platform. In Figure 7 the migration time increases almost linearly when the agent size is more than 3 MB, but a different behavior can be observed in the other situations.

4.2 Message Plugin

Here we are going to evaluate the message passing capability of the Jade platform. Some significant indexes can be found in literature [4,5]: Average Round Trip Time (aRTT), Spacing time, Receiving time, Processing Time. We chose to evaluate the Round Trip time $T_r$ defined as the time necessary to exchange an ACL message between an agent sender and a set of agent receivers following a round trip. It means that the message must be sent from the first agent and it must be forwarded from the second agent to the next one till it is returned to the original sender. The parameters affecting the experiment are:

- $\text{dim}$: message size;
- $N$: number of agents;
- $P(Ag_{send}, Ag_{rec})$: location of agents in the platform.

$$T_r = T_r(\text{dim}, N, P(Ag_{send}, Ag_{rec}))$$
Also here the message size has been chosen big enough in order to make trivial the effect of the operative system on the measures. About the agents’ location we must observe that messages can be exchanged according three different communication mechanisms:

1. by an event, if sender and receiver execute on the same container;
2. by RMI (Java Remote Method Invocation), if agents execute on different container of the same platform;
3. by IIOP or http, if agents execute on different containers of different platforms.

Of course if agents execute on different nodes also latency, bandwidth and traffic of the network affect the estimated index. The provided results have been obtained creating agents on different containers of the same platform. Each container executes on a different node and the only traffic is generated by communication among agents. Figure

```xml
<Experiment name="ScambioMessaggi">
  <AgentConfig name="AgeConf" class="plugin.confAgMigr5" />

  <TMAgent name="AgenteTestMan" AgentTMClass="plugin.testManaAgt5"/>
  <TestSet IDt="N1">
    <repetitions>30</repetitions>
    <messageSize>128</messageSize>
  </TestSet>
</TMAgent>

<AgentTest name="AgenteSend" class="plugin.SenderAgent" />
<AgentTest name="AgenteRec" class="plugin.ReceiverAgent" />

<DCAgent name="AgenteRD" class="plugin.datascolAgtMess">
  <Params>
    <Output filename="MessageOutput.xml" format="xml" />
  </Params>
</DCAgent>
</Experiment>
```

Figure 8. Configuration file for Message Passing

8 shows an example of plug-in configuration. Two TestAgent will be created and, in the first test, the round trip time will be evaluated by 30 exchanges of 128KB messages Figure 9 show performance figure of communication between two agents. Different configuration have been evaluated: agents execute in the same container (1 node - 1 container), in different containers on the same node (1 node - 2 container, that means different jvm), in different containers on different nodes (2 nodes - 2 containers). Agents’ containers are always different from the Main-Container in order to not let the results be affected by the execution of other agents belonging to the framework or to the agent platform. Message size varies from 1 MB to 10 MB. As we can see the round trip time increases with the message size even an irregular behavior can be observed when the size is 5MB. Better performances are obtained when the agents run on the same container and messages are passed by event in the same jvm.

![Figure 9. Performance of Message Passing](image)

Figure 9. Performance of Message Passing

5 Conclusions

We argued in this paper about the relevance of performance analysis of mobile agent platforms. Actually there are not wide accepted benchmark or performances indexes which allow to characterize and compare different agent frameworks. We presented a tool to support the development and execution of benchmarks for performance evaluation of mobile agents based distributed systems. In the proposed approach tool and the benchmark are implemented by software agent themselves. We presented the software architecture, and early implementation tested by two case studies. Mobile agent represent mobile benchmarks which can move across the network to perform performance measures execution on the remote host. We provided experimental results which show an unexpected behavior of migration and communication facilities in the Jade platform. Additional experiments and analysis need to understand the detected behaviours. Our contribution represent a first step for further investigations aiming at defining effective indexes and benchmark for performance
evaluation of these kinds of platforms.

Acknowledgments
We are particularly thankful to Ing. Luigi Pontillo for his technical support.

References