Using Mobile Agents for Parallel Processing

Luís Moura Silva, Victor Batista, Paulo Martins, Guilherme Soares

University of Coimbra
Dep. de Engenharia Informática
Pólo II, 3030 Coimbra, Portugal
Email: luis@dei.uc.pt

Abstract
Mobile code is a promising model for distributed computing and it has been exploited in several areas of applications. One of the areas that may benefit from the use of mobile agent technology is parallel processing. This paper describes a Java-based platform, called JAMES, that provides support for parallel computing. We have implemented a software module that supports the well-known model of Master/Worker and we have exploited the use of parallel computing in some distributed tasks. We will present some experimental results that compare the Master/Worker model with the usual model of migratory agents. Then, we compare the use of mobile agents with two other solutions for parallel computing: MPI and JET. The results achieved are quite promising.

1. Introduction
Mobile code is an emerging paradigm that is now gaining momentum in several fields of applications [1]. A mobile agent corresponds to a small program that is able to migrate to some remote machine, where it is able to execute some function or collect some relevant data and then migrate to other machines in order to accomplish another task. The basic idea of this paradigm is to distribute the processing through the network: that is, send the code to the data instead of bringing the data to the code.

When compared with traditional client/server solutions mobile agents can potentially reduce the traffic in the network, provide higher scalability, more flexibility and robustness. In the last few years the use of mobile agent technology has received an extraordinary attention from several Universities and research institutes and a notable investment from leading companies [2]. Mobile agents have been applied in several areas, like mobile computing, electronic commerce, Internet applications, information retrieval, workflow and cooperative work, network management and telecommunications [3][4][5][1]. Several commercial implementations of mobile agents have been presented in the market, including Aglets from IBM [6], Concordia from Mitsubishi [7], Odyssey from General Magic [8], Voyager from ObjectSpace [9], Jumping Beans from AdAstra [10] and Grasshopper from IKV [11].

The type of applications that are most appropriate for mobile agent technology would include at least one of the following features: data collection, searching and filtering, distributed monitoring, information dissemination, negotiating and parallel processing [12]. When the application is computationally intensive or requires the access to distributed sources of data then the parallel execution of mobile agents in different machines of the network seems to an effective solution. At the same time, the dynamic and asynchronous execution of mobile agents fits very well in changing environments where it is necessary to exploit some notion of adaptive parallelism.

Most of the existing agent platforms still lack the necessary support for parallel computing. This paper presents a Java-based Mobile Agent infrastructure that includes programming support for one model of parallel processing that is well-suited for the execution of mobile agents: the Master/Worker model. In this model, the application is partitioned into several independent tasks that can be executed in parallel by a set of mobile agents. There is a special agent – the master - that manages the distribution of the tasks and the collection of the results. The other agents
– the workers - are sent to the network to perform the real computation.

This module of parallelism was implemented in the JAMES platform, a Java-based mobile agent infrastructure that has been developed on behalf of a Eureka Project (Σ1921) by three project partners: University of Coimbra (Portugal), Siemens SA (Portugal) and Siemens AG (Germany). The JAMES platform is mainly oriented for applications in the area of Telecommunications and Network Management but it also provides support for the execution of traditional parallel applications.

The main motivation of our study is twofold:
(i) realize how the mobile agent technology can be used to execute the traditional applications from the HPC community (e.g. number-crunching, numerical applications and event-driven simulations);
(ii) apply the concepts of parallelism in the execution of mobile agents in some distributed applications, like performance management and data collection in systems of telecommunications.

This paper presents some preliminary, but insightful, conclusions about the functionality and the performance of parallel execution in a mobile agent system. These results can also be applied to other similar platforms.

The rest of the paper is organized in the following way: section 2 presents an overview of the JAMES platform. Section 3 describes the models of execution of mobile agents while section 4 explains in some detail the platform support for the Master/Worker model. Section 5 refers to other platforms that have some support for parallel processing and section 6 presents some experimental results. Section 7 concludes the paper.

2. An Overview of the JAMES Platform

The main goal of the JAMES project is to develop an infrastructure of Mobile Agents with enhanced support for network management and we try to exploit the use of this new technology in some software products of telecommunications.

The existing applications in the management of telecommunication networks are usually based on static and centralized client/server solutions, where every element of the network sends all the data to a central location that executes the whole processing over that data and provides the interface to the user operator. By these reasons, the resulting applications usually have problems of scalability, software maintenance and produce too much traffic in the network.

The use of mobile agents in this kind of applications represents a novel approach and potentially solves most of the problems that exist in centralized client/server solutions. The applications can be more scalable, more robust, can be easily upgraded and customized, they reduce the traffic in the network. All this is achieved by distributing the processing functions over the network through the use of mobile agents instead of relying on a centralized processing solution.

The JAMES project will try to exploit all these technological advantages and see how the mobile agents technology can be used in software products that are been developed by Siemens SA. The project involves the development of a Java-based software infrastructure for the execution of mobile agents. This platform is mainly oriented to the applications in the area of telecommunications and network management. The main goals of the platform can be enumerated as follows: high-performance, security, fault-tolerance and robustness, support for network management, easy upgrading, disconnected operation, a 100% pure Java implementation and the integration with CORBA services.

The JAMES Platform provides the running environment for mobile agents. There is a distinction between the software environment that runs in the manager host and the software that executes in the network elements: the central host executes the JAMES Manager while the nodes in the network run a JAMES Agency. The agents are written by application programmers and will execute on top of that platform. The JAMES system will provide a programming interface that allows the full manipulation of mobile agents. Figure 1 shows a global snapshot of the JAMES system.

The host machine that runs the JAMES manager is responsible by the whole management of the mobile agent system. It provides the interface to the end-user, together with a Graphical User for the remote control and monitoring of agents, places and applications. A snapshot of this interface is presented in Figure 2. The James
GUI is the main tool for management and administration of the platform. With this interface, the user can manage all the Agents and Agencies in the system.

Every Network Element runs a Java Virtual Machine and executes a JAMES Agency that enables the execution of the mobile agents. The JAMES agents will migrate through these machines of the network to access some data, execute some tasks and produce reports that will be sent back to the JAMES Manager. There is a mechanism of authentication in the JAMES Agencies to control the execution of agents and to avoid the intrusion of non-official agents. A special agent migration protocol was developed to transfer the agents across the machines in a robust way and to assure the atomicity in the occurrence of failures.

The main features of the JAMES platform can be summarized in the following list:

- Kernel of the JAMES Manager and JAMES Agency;
- Service for remote updating of Agents and Agencies;
- Atomic migration protocol;
- Support for fault-tolerance through checkpoint-and-restart;
- Reconfigurable itinerary;
- Support for disconnected computing;
- Watchdog scheme and system monitoring;
- Mechanisms for resource control;
- Flexible code distribution (caching and prefetching schemes);
- Logging and profiling of agent activity;
- GUI interface to allow the remote control of agents;
- Integration with CORBA services;
- Integration of a Java-based SNMP stack;
- Multi-paradigm for agent execution.

The explanation of all these mechanisms is out of scope of this paper, but in the next section we will describe in some detail the different execution models that are provided by the platform.

3. Models of Execution

The JAMES Platform will support three main paradigms for agent execution:

(i) Simple mobile agent
(ii) Migratory agent
(iii) Master/Worker model

3.1 Simple Mobile Agent

In this case the JAMES Manager sends a single agent to a particular Agency of the network. This agent will perform a set of local interactions with the resources of the JAMES Agency. After performing all the operations it sends back a report containing the final result. Figure 3 presents the typical execution of this type of agents. In practice, these agents are only allowed to execute push/pull operations.
This model compares directly to the RPC (Remote Procedure Call) model, and it is advantageous to use when the agent has to perform several interactions with the Agency, thereby avoiding remote communication and reducing the network traffic. When the interaction is scarce the system can make use of simple message communication (RPC-like). This trade-off decision depends on the application characteristics, the latency of the network and the frequency of agent execution. Some experimental study should be performed to collect some data from the system in order to provide some support for this decision.

This simple agent should be mainly used when the Manager only needs some information from a network element and does not require any global vision of the network.

The Jumping Beans platform [10] only supports this paradigm of execution: an agent is only allowed to travel directly from a central server to an agency. If the agent wants to migrate between two Agencies it has always to move through the central server. This may represent a serious bottleneck in the execution of the applications.

3.2 Migratory Agent
A migratory agent is a mobile agent that travels around several Agencies of the network. The list of nodes that is visited by the mobile agent is specified by a Itinerary object. In each Agency the agent executes a particular mission, which corresponds to a set of local tasks. This sort of agent will be used when the overall task requires a global report of the network. The previous Figure 1 depicts the typical execution of a migratory agent.

Apart from Jumping Beans, all the other mobile agent platforms support this model of execution.

3.3 Master/Worker Model
In this paradigm there is a master agent that executes in the JAMES Manager. The master creates a set of worker agents and sends them to the Agencies of the network. Every mobile agent will work in parallel and execute a set of tasks in a different Agency. The platform provides support for agent replication, inter-agent communication and collective operations. Figure 4 presents an example of this type of agents.

We envisage the use of parallelism with mobile agents in two different scenarios:
(i) for instance, there are some distributed computations that require a high CPU load. These applications can be broken into independent tasks and be delegated to a pool of agents;
(ii) other applications may require data collection or information filtering from a set of machines of the network. This sort of applications would also benefit from the parallel execution of mobile agents.

In the next section we will present a programming example of a Master/Worker application using mobile agents in JAMES. There is one assumption associated to this model: the tasks that are executed by the worker agents should be independent in order to be executed in parallel.

4. The Master/Worker Model: A Programming Example
To best illustrate the functionality of this we will present a simple programming example.
The Master/Worker API provides two different flavours: the synchronous and the asynchronous model. In the first model, the master launches the pool of agents and then waits for the atomic execution execution of all the workers. In the asynchronous model, the master can collect some partial results from some worker agents and do some concurrent computation with the rest of the agents. This second model also includes methods for agent communication, event listeners, collective operations and support for load-balancing.

For the sake of simplicity we will present an example that makes use of the synchronous model. The code is shown in the next three Figures. In this program the Master agent creates three Worker agents that will be launched in three remote sites, known as Agency1, Agency2 and Agency3. These Workers have a very simple task to do: they will create a random number, and send it back to the Master. The Master will collect the results and print them in the standard output.

The code of the master agent is presented in Figure 5.

```java
// MasterExample.java
import james.MasterInterface;
import james.AgentResults;
import james.Master;
import james.Agent;
import java.util.Arrays;
public class MasterExample extends Agent implements MasterInterface {
    int total = 0;
    public void master() {
        Master master = new Master(this);
        String[] hosts = new String[3];
        hosts[0] = "Agency1";
        hosts[1] = "Agency2";
        hosts[2] = "Agency3";
        master.launchWorkers("WorkerExample", "methodToCall", hosts, null);
        workResults(master.waitForWorkerResults());
    }
    public void workResults(AgentResults[] arrayOfResults) {
        for (int i = 0; i < arrayOfResults.length; i++) {
            AgentResults ar = arrayOfResults[i];
            WorkerResults workerResults = (WorkerResults) ar.getData();
            total += workerResults.getResult().intValue();
            println("Agency " + workerResults.getAgency() + " computed the value " + workerResults.getResult().intValue());
        }
        println("WorkResults - The total is : " + total);
    }
}
```

Figure 5: The code of the master agent.

As can be seen in the Figure, the master agent extends the class james.Agent. Any agent can be a master: it only needs to create an object of type james.Master and this object will export all the API of the master. An agent can be simultaneously a master and a worker. This way we can easily use this model to implement a hierarchy of master and worker agents, when there is a need for a scalable solution.

In this example, the first thing the master does is to create the Master object. After that, it launches the worker Agents. This is done through the following method:

```java
master.launchWorkers("WorkerExample", "methodToCall", hosts, null);
```

This method indicates the name of worker agents, the method they should execute in the remote sites and a list of Agencies to where the agents should be dispatched.

After that, the master agent waits for the results computed by the workers, by calling the method:

```java
master.waitForWorkerResults();
```

This method will only return when all the results become available and it returns an array of objects of type WorkerResults. The JAMES API provides other non-blocking methods or the possibility to block only for a limited amount of time.

The next Figure presents the object that is sent by workers to the master agent. This class should implement the Serializable interface.

```java
// WorkerResults.java
import java.io.Serializable;
public class WorkerResults implements Serializable {
    private String agencyName = null;
    private Integer result = null;
    public WorkerResults(String agencyName, Integer result) {
        this.agencyName = agencyName;
        this.result = result;
    }
    public String getAgency() {
        return agencyName;
    }
    public Integer getResult() {
        return result;
    }
}
```

Figure 6: The code of the result object.

It is not mandatory to define a specific object to represent the worker results. Any native or existing Java object can also be used instead. For instance, in this case we could have use an Integer object and it would have not been necessary to implement this class (WorkerResults).
In the next Figure we present the code of a Worker agent. The first step is to create an object of type `james.Worker` thereby enabling access to the Worker API. The task to be executed by each agent is defined inside the method: `methodToCall()`. After executing its (simple) task the Worker creates an object to send the result to the Master. The method `getCurrentAgency()` returns the name of the current Agency where the agent is running on. The result is effectively sent to the Master by calling the following method: `worker.sendResult(myResult).

```java
// WorkerExample.java
import james.Agent;
import james.Worker;
import james.Mission;
import java.util.Random;
public class WorkerExample extends Agent {
    int count = 0;
    public void onCreation(){
    }
    public void methodToCall(){
        Worker worker = new Worker(this);
        Integer result =
            new Integer(new Random().nextInt());
        WorkerResults myResult =
            new WorkerResults(getCurrentAgency(),
                result);
        println("WORKER - my result is "+ result.intValue());
        try{
            println("\tWORKER - Sending results to Manager");
            worker.sendResult(myResult);
        } catch (JamesResultsException e){
            System.out.println("Error sending results to manager!");
        }
    }
}
```

Figure 7: The code of the worker agent.

This code presents a very simple and synthetic application just to give a broad overview about the Master/Worker synchronous API. The interested reader should consult the documentation and the User’s Guide of the platform to get more details about the rest of the API.

5. Related Work

Some of the other platforms provide support for group collaboration and agent cloning. The Concordia platform [13] allows the creation of a group of agents that can work cooperatively in some distributed task. There is some support for agent collaboration and communication but it still lacks special support for parallel computing. The Aglets environment [14] provides special support for a Master/Slave design pattern, where there is a master agent that can delegate a task to a single slave agent.

The platform provides support for communication between these two agents. This pattern would allow some concurrent execution of tasks but could be more extended in the support for parallelism.

There are two other mobile agent systems that have been proposed for parallel computing: Wave [15] and Messengers [16]. Both systems provide the notion of autonomous objects that can roam around the network and execute computational tasks. However, these systems were not implemented in Java and have restricted portability. The programming API also lacks some high-level constructs for parallel computing.

It would be possible to exploit the parallel execution of mobile agents in any of the other existing platforms, although with some effort from the application programmer. The JAMES platform provides a high-level API that simplifies the task of the programmer and provides a scalable solution for the use of Master/Worker agents.

6. Some Experimental Results

In this section we will present some experimental results that were collected when using mobile agents in two different scenarios: first, in a distributed search engine; secondly, in two typical parallel benchmarks (I calculation and Linpack). The benchmarks are meant to illustrate two possible situations where mobile agents can be deployed.

6.1 Parallel searching engine

In this application there is a set of machines in the network and each of these machines has a text file with some information. The goal of the application is to find the occurrence of a specific string inside those distributed files. For this purpose we make use of mobile agents for doing that distributed search operation. The idea of this application is to resemble a typical scenario of information searching in Internet or Intranet environments.

We have implemented this program using the three models of execution: simple agent, migratory agent and the Master/Worker model. The application was executed in a cluster of 6 machines running Microsoft Windows NT and JDK 1.1.5. One of the machines acted as the JAMES Manager and the Agencies were installed in the other five. These machines are connected through a 10Mbits/sec Ethernet switch. The experiments were conducted when
the machines and the network were fully dedicated and the presenting results presented correspond to the average of at least 3 different executions. The variance across different executions was not significant.

Figure 8 presents the execution time (in seconds) when the application had to search through a set of 5 files, each one with a size of 1Mbytes. The internal size of the mobile agent was increased from 1Kbytes, 100 Kbytes and 1Mbytes. This variation was introduced only to observe the relevance of the agent’s size in the individual experiments.

Figure 8: Execution time with a file size of 1Mb.

In this first experiment the size of the file is relatively small and the execution time is still mostly reflected by the migration time. Figure 8 shows that when we increase the size of the agent the simple mobile agent performs worse than the other two models. The simple agent is only allowed to do push and pull operations between an Agency and the central server. The Jumping Beans platform only provides this model. So, we can conclude that providing a decentralized migratory solution would clearly improve the performance of the applications. The execution time of the migratory agent and the Master/Worker model is very similar but as can be seen the parallel execution still brings some improvements.

If we increase the size of the files to be searched then we also increase the level of computation, and this is the case where parallel computing starts to pay-off. In Figure 9 we present the same application but this time the size of the distributed files was increased up to 1Gbytes. In the overall this corresponds to an intensive search in 5Gbytes of text data. This time, the computation time dominates the overall execution time.

The results achieved show that there were no significant differences between the simple and the migratory agent since both models perform a sequential search and the migration time is really small when compared with the computational time. The most interesting fact is the improvement that was achieved when we used the Master/Worker model. This improvement corresponds to a speedup of 4.92. This result was fairly expected: since the search operations were not correlated they could have been done in an independent way and the speedup achieved was almost linear.

Figure 9: Execution time with a file size of 1Gbytes.

These results clearly show that parallel computing can be highly effective in typical applications of mobile agents when there is a large volume of data to compute. The Master/Worker model is easy to use and brings some substantial benefits in these situations.

6.2 Parallel execution of numerical applications

In this section we have done some experiments with the use of mobile agents in numerical applications. We used two well-known benchmarks: the Pi calculation and Linpack. Both programs were executed as embarrassingly parallel applications, that is, each worker process executed independently of each other the sequential version of the algorithm. This model would resemble some parallel decomposition of the input data.

In order to increase the computational level of each task the Pi calculation was executed with 10,000,000 hits and the Linpack program has used a problem size of 1000 equations. Both applications were implemented in Java and used the Master/Worker API of the JAMES platform. The program was also executed in the same cluster of 5 Agencies and using the same dedicated conditions.

Our goal was to compare the performance of the mobile agents with the traditional solutions for parallel computing in clusters of
workstations. For this purpose, we have also implemented those two benchmarks in two other environments:

(i) **Java-WMPI**: an implementation of MPI for Windows NT [17] that has a Java programming interface. The Java bindings are described in [18];

(ii) **JET**: a software infrastructure that supports the parallel processing of CPU-intensive problems by using Java applets. This tool exploits the idea of volunteer computing and is mainly targeted to Web-based environments. More details can be found in [19].

MPI is the standard for parallel computing and has the most comprehensive support for parallelism. While JAMES and JET are only restricted to Master/Worker applications MPI can be used in any sort of application. The JAMES platform does not intend to compete with MPI; we are only comparing its behaviour in the execution of some parallel applications.

The distribution of code in these three environments is significantly different: MPI requires the existence of a shared file system or the pre-installation of the application binaries in every machine of the network. JET is based on the execution of Java applets that executed inside a Web browser in the client machine. Applets are only able to push operations and are restricted to the Web security sandbox. Finally, in JAMES the code of the Worker agents is migrated from the central machine (Manager) to the remote sites (Agencies). The upgrading and the management of remote software in mobile agent systems is much more flexible than in MPI.

In Figure 10 we present the execution time of those two applications when executing in these three environments.

For the Linpack application there were no significant differences and all the three tools presented similar execution times. This is a computational intensive application that mainly performs floating-point operations.

In the case of the Pi calculation the results were rather different: the execution of the Java applets in the JET environment performed worse than the other tools (almost the double of the execution time). This result was not expected and we had to investigate a bit further the reasons for this behavior. After some small experiments we realized that it had to do with the low performance of the Netscape JVM in the execution of synchronized methods. This particular application makes extensive use of Math.random() that is a synchronized method. We replaced this method by a similar one (non-synchronized) and the performance of the application when executing with JET was similar to the performance of JAMES.

In the comparison between JAMES and MPI the parallel execution of mobile agents was fairly better. Both results were collected in the same machines and in the same conditions of the network. We have still observed an interesting fact: the total time of the MPI version took 16.4 seconds while the time spent in the Pi algorithm was only 13.06 seconds, quite close to the 13.579 seconds that were measured with the JAMES platform. The difference of 3 seconds has to do due with the initialization of MPI through the network. Apparently, the execution of the MPI_Init() function is rather inefficient since it requires the launching of MPI remote daemons and application processes in those machines of the cluster. The initialization of the mobile agents in the JAMES platform was much more efficient.

The performance of JAMES seems to be very competitive when compared with MPI and JET. Nevertheless, there are some other relevant features connected with the execution of mobile agents that can bring more benefits to the applications than the two other tools, namely:

(i) mobile agents provide more robustness against failures;

(ii) they are easy to upgrade and customize;

(iii) they are more flexible to adapt to changes in the network;

(iv) they allow for asynchronous computation and fit well in situations of disconnected computing.
These advantages can be of paramount importance in some complex distributed applications that execute in heterogeneous environments and need to adapt frequently to changes in the network and in the software. The Master/Worker API of the JAMES platform does not provide the same functionality of MPI but it represents a simple and effective solution to the exploitation of parallel computing with mobile agents.

7. Conclusions

This paper presents a Java-based mobile agent platform that has support for parallel computing. This platform provides a high-level programming API for one common design pattern: the Master/Worker model. The applications that can make use of this API should be easily broken into separated tasks that are executed in a parallel by a pool of mobile agents that are sent to the network. This Master/Worker model can be used in traditional numerical parallel applications and in distributed tasks, like information filtering and data collection, that may require some parallelism in the execution.

We have compared the performance of the Master/Worker model with two other common models: the simple push/pull mobile agent and the traditional migratory agent. Then, we compared the behavior of JAMES with two other tools: MPI and Applet-based parallel computing. The results achieved were very promising and some preliminary conclusions could be taken related with the use of mobile agents for parallel computing.

The first version of the JAMES platform is already finished and we are now doing a benchmarking study to compare the use of mobile agents over traditional client/server solutions to see if we corroborate some of the advantages of this new paradigm in the field of distributed computing.

Acknowledgments

This project is partially supported by Agência de Inovação and was accepted in the Eureka Program (E!1921). Many thanks to our project partners by providing us with some important suggestions for the implementation of the platform.

References