MAGDA: a software environment for Mobile AGent based Distributed Applications *

R. Aversa, B. Di Martino, N. Mazzocca, S. Venticinque
Facoltà di Ingegneria - Dipartimento di Ingegneria dell’ Informazione
Second University of Naples - Italy
{aversa,venticinque}@unina2.it, {beniamino.dimartino,n.mazzocca}@unina.it

Abstract

The Mobile Agents model has the potential to provide a flexible framework to face the challenges of High Performance Computing, especially when targeted towards heterogeneous distributed architectures. We developed a framework for supporting programming and execution of mobile agent based distributed applications, the MAGDA (Mobile AGents Distributed Applications) toolset. It supplements mobile agent technology with a set of features for supporting parallel programming on a dynamic heterogeneous distributed environment.

1 Introduction

The Mobile Agents model [2] represents an effective alternative to the Client-Server paradigm in several application fields such as e-commerce, brokering, distributed information retrieval, telecommunication services [1]. Mobile agents have the potential to provide a flexible framework to face the challenges of High Performance Computing, especially when targeted towards heterogeneous distributed architectures. Several characteristics of potential benefit for scientific distributed computing can be provided by the adoption of the mobile agent technology, as shown in the literature [3, 4, 5]: they range from network load reduction, heterogeneity, dynamic adaptivity, fault-tolerance to portability to paradigm-oriented development. Mobile agents meets the requirements of the heterogeneous distributed computing since the agents are naturally heterogeneous, they reduce the network load and overcome network latency by means of the mechanism of the migration, they adapt dynamically to the computing platform through the migration and cloning, and finally the different tasks of a sequential algorithm can be embedded into different mobile agents thus simplifying the parallelization of the sequential code.

We are developing a framework for supporting programming and execution of mobile agent based distributed applications, the MAGDA (Mobile AGents Distributed Applications) toolset. It supplements mobile agent technology with a set of features for supporting parallel programming on a dynamic heterogeneous distributed environment.

The (prototype) framework currently supports the following features:

- collective communication among mobile agents, with provision of a set of collective communication primitives;
- dynamic workload balancing, through centralized or distributed mechanisms;
- dynamic system parameters estimation for each node of the cluster;
- automated mechanisms for agents’ migration and cloning, driven by policies based on system monitoring, in terms of utilization, idle time, memory usage and different run-time events (like shutdown);
- Agents’ authentication, which can be used in security critical applications;
- remote agents’ creation, which provides the possibility to activate an agent from different terminals, including wireless handheld devices;
- a skeleton-based parallel programming environment, based on specialization of Skeleton Java interfaces and classes;

*This work has been supported by the Italian Ministry for University and Research (MURST) (P.R.I.N. Project ISIDE - “Dependable reactive computing systems for industrial applications”) and by the CNR - Consiglio Nazionale delle Ricerche, Italy (Agenzia 2000 Projects METODOLOGIE E STRUMENTI PER LABORATORI VIRTUALI DISTRIBUTI and ALCOR.)
• integration of MA programming paradigm and OpenMP, for programming hierarchical distributed-shared memory multiprocessor architectures, in particular heterogeneous clusters of SMPs (and uniprocessor) nodes.

We based the development of our framework on the basic agent mechanisms provided by the Aglet Workbench, developed by IBM Japan research group [1]. An aglet (agile applet) is a lightweight Java object that can move to any remote host that supports the Java Virtual Machine. An Aglet server program (Tahiti) provides the agents with an execution environment that allows for an aglet to be created and disposed, to be halted and dispatched to another host belonging to the computing environment, to be cloned, and, of course, to communicate with all the other aglets.

In the following sections we present a description of the features provided by the MAGDA framework, mainly in terms of functionality, with some insights in the design and implementation choices.

In [7] a case study is presented, where the framework is utilized to program a parallel combinatorial optimization performed with Branch & Bound (B&B) technique.

2 Location Transparency and Collective Communication

In a distributed environment, especially when the Mobile Agent programming paradigm is adopted, a relevant feature which a framework should provide to the programmer is the visibility of the network configuration and a location transparent communication system.

The agent should be able to explore the network and to look for the active servers to be hosted, and to communicate with the other agents without to know where they are physically running. Finally in a highly dynamic computing platform that represents the target computing environment of most mobile agent based applications, it becomes essential for the user to benefit of some collective mechanisms of communication and synchronization.

The only collective communication primitive supported by the Aglet Workbench is the multicast primitive, but only within a local context: an aglet that requires to receive a multicast message with a specific label needs to subscribe to that kind of multicast message.

The original IBM Aglets Workbench provides the programmer with a proxy object, which is a reference to an agent, in order to allow point to point communications. The proxy becomes invalid when the agent disposes or moves it-
self to an other host. So the point to point communication is still location dependent because the framework does not automatically update the agent proxy in order to preserve the agent reference. Another problem deals with the possibility of exploring the network looking for active servers.

In order to solve this kind of problems we have designed a software protocol which is employed to build a regular binary tree topology made of the active servers.

All the information that characterizes the topology of the agent servers is dynamically updated by means of enrol and disjoin procedures and is maintained in a suitable data structure stored in a permanent node server.

Each new computing node joining the agent framework contacts the permanent node server. It takes care of providing the new node with the identity in the topology and with the reference to the father node; in addition it notifies the father node about the changes in the structure.

The knowledge of the topology by the server, even in a local area, provide the agent with the possibility to explore the network moving itself among neighbour nodes. On the other side the server is able to forward broadcast and multicast messages across the web of active servers.

Relying on such topology we were able to augment the framework with collective communication primitives such as broadcast and multicast, and some collective synchronization operations that, for example, allow to compute and distribute a maximum or minimum value among all the involved agents.

Other examples of features can be designed and tested. For example we are working about the automatic relocation of the agents in the network for location transparent point to point communication.

Just to exemplify we illustrate the implementation of the multicast primitive that is based on a communication server running on each host. Such server performs the following actions:

- waits for a multicast message labeled "remote multicast";
- captures in a local context a "remote multicast" message;
- forwards a copy of the multicast message to the neighbour hosts in the topology.

Each neighbour communication server on the remote message reception:

- sends in a local context the multicast message with the original label, so allowing any subscribed agent on that host to receive the message;
- forwards the message to its neighbours if it have

To summarize, an agent that wishes to send a remote multicast, must perform the "remote multicast" primitive by means of the specific primitive and providing the message with the subject label, while an agent interested to receive multicast messages has to subscribe to a multicast message with the specific subject label.

The user is able to choose different multicast algorithms, for example any multicast message can be sent to master server (the root of the tree), and then it is forwarded from each father node to the sons till to reach the leaves of the tree.

### 3 Dynamic workload balancing

We provide a service for dynamic workload balancing that can be easily customizable to any user application developed within the Workbench.

The framework is composed of a coordinator agent which controls the load balancing and an Aglet superclass support, this latter must be inherited by the user class.

The coordinator agent communicates, by message passing, only with the Aglet superclass. In order to use the support, the user class must override some methods inherited by the superclass and set some variables. The implementation of these functions depends on the specific user application.

The coordinator manages a first list of registered workers and a second list of available free hosts. When the user Agent's execution starts, the coordinator is created and executes a polling interaction with the working agents registered with it, in order to know the state of their computation, that is the percentage of computation performed, with respect to the computation amount assigned to it. The registered agent's references are stored in a vector, ordered according to their computation state; the ordering of this vector thus represents the relative speed of each worker with respect to the others. It also gives a representation of the state of the computation.

A load unbalance event occurs when:

1. a worker ends the computation assigned to it, and becomes idle;
2. the slowest worker has completed a percentage of the computation amount assigned to it which is far below the percentage completed by the fastest worker (determined by a fixed threshold on the difference of the two percentage).

In the first case, the idle worker notifies the coordinator of its availability to receive workload; the coordinator then asks the slowest worker (according to the vector of computation states) to send a portion of its workload to the idle worker.
In the second case, the following two situations can occur:

1. the list of available hosts is not empty: the coordinator then creates a generic slave agent and dispatches it to the target machine, then asks the slowest worker to send a part of its load to the slave.

2. the list of available hosts is empty: the coordinator then asks the slowest worker to send part of its load to the fastest worker.

The coordinator periodically updates the vector of the computation states, by executing a polling at a fixed, user-defined, frequency.

In the following some more details are provided on how the workload is represented and exchanged. Part of load is sent from an agent to another by means of a message carrying an object belonging to a class defined by the user (UserWorkload), implementing the Serializable (which is a standard Java interface) and the Load interface. The data declared in class UserWorkload represent the load and the information to elaborate it, while the method exec of the Load interface defines the way to elaborate the data. Such a method is called by the receiver to perform the computation. The user can define different kinds of UserWorkload classes, according to the kind of computation to send (in different points of the program we can treat different problems) or according to the destination agent (slave needs all data and code, while worker can reuse its code or its data).

Finally we describe the actions to be performed by the user of the framework, in order to specialize the balancing algorithm and to grant its correctness. Some methods have to be overridden, in order to specify how:

- to calculate and communicate the percentage of work done;
- to create and prepare the load assigned to the fast/idle worker or to the slave;
- to specify if the fast/idle worker or the slave must return the result of its computation;
- to elaborate the results eventually received from the fast/idle worker or the slave.

To personalize the behaviour of the coordinator the user must provide the following information:

- the frequency rate of the polling for updating the vector of computation states;
- the threshold difference between the percentages of computations done which triggers the load transfer among the workers;
- all the URLs of the hosts of the cluster.

To personalize the behaviour of the workers the user can set some variables inherited from the superclass support. In particular he can choose:

- if a worker wants to receive any load from the others;
- if it needs to be given back any result possibly computed by the workload passed to another worker agent.

### 4 Skeleton Based Mobile Agent Programming

Using the mobile agent paradigm remains a non trivial task because the mobility feature introduces additional difficulties in designing coordination, synchronization and communications among the different tasks. In most cases, especially when the starting point is an available sequential code, the use of algorithmic skeletons can ease the programming task, and the mapping for performance, on parallel systems. We have defined and implemented a set of Java packages, which enable to program distributed applications by adopting a skeletons-like approach, exploiting the peculiar features of both Object Oriented and Mobile Agents programming models. By means of the provided skeletons interfaces the programmer could be able to implement its own application by specialising an assigned structure and utilizing the set of functionalities that the mobile agents framework offers. In addiction such approach allows to reuse a great deal of the sequential code, when available. A predefined algorithmic skeleton allows to follow the sequential programming model by filling some methods, classes and interfaces and to hide the difficulties involved by an explicit parallel programming paradigm. The difficulties and the features of the Mobile Agent programming paradigm can be managed at a lower level, transparent to the user.

Two algorithmic skeletons involving the Farm-like programming paradigm and the Divide and Conquer-like programming paradigm respectively have been implemented and tested [21]. In Processor Farm programming paradigm the master process create a number of slaves and assigns some work to everyone of them. The slaves compute their work and return the results to the master. Task Queue is the most general Farm-like skeleton, every slave may produce new work to be performed by itself or by other slaves. The second algorithmic skeleton we have implemented belongs to Divide and Conquer-like skeleton class, but not to the highest abstraction level. It is an example of Tree computation algorithmic skeleton. It solves the initial problem dividing it in several subproblems assigned to different agent workers. The data flow from the root into the leaves and
the solutions flow back up towards the root. We have chosen to implement a binomial algorithm to build our tree, so its shape and the results recombination procedure is consequentially determined.

5 System Activity Monitoring

The MAgDA framework provides primitives which allow a dynamic monitoring of system parameters such as the CPU and memory utilization. A further significant performance metric is the available bandwidth of the network that is useful to compute the cost of transferring the data and the code from an host to another. A set of Java api has been developed to collect periodically these parameters values by the servers which run on identified operative systems such as Linux and Solaris system. The MAGDA server uses Java runtime system to employ operative system calls which help to estimate the system status. A monitoring thread of major system activity and resources allows the programmer to design some balancing strategies to optimize the system utilization and throughput. The values of interest are stored in a table that can be queried by each agent who wants to know this kind of information. In order to obtain a global knowledge of the system activity current status, a dedicated traveling agent could visit the servers of the cluster and collects all the local system information. The collected information can be used to optimize the distribution of the application workload among the agents or to move an agent from a heavier loaded computing machine to a less loaded one.

6 Remote Agents’ Creation

A set of Java api’s, which can be extended and specialized by the programmer, has been developed to provide the Mobile Agent platform with the possibility of starting an agent from a remote terminal. The protocol used to ask the server for the agent’s creation is http. This choice allows the user to use even a web enabled phone in order to initialize and start the desired application. The implemented api set was tested by a Java midlet, developed within the JAVAME development toolkit. The user is able to download the midlet for the required application; by means of the midlet the connection to the agent server and the initialization of the chosen application is automatically performed. The midlet provides an option to download the results to be delivered from the Agent Server. The handheld device used to test the midlet is the Motorola Accompil (GSM/GPRS javaphone). The user is able, when it needs, to authenticate, by means of Java library dealing with digital signature technology, the collected results.

7 Agents’ Authentication

Security problems are critical in distributed computing especially when the system used is geographically distributed and the computing paradigms employ the code mobility. The IBM Aglets workbench allows the administrator to design security policy in order to grant or deny to the agents some services such as dispatching, cloning, file system reading or writing, socket opening ... The light mechanism of jar signer can be applied to the code when it is stored in a single jar archive, but about certification of the keys and privacy nothing is done. Finally such security function is referred to the agent code and not to the value of its data at the dispatching and recovering time. Authentication of a server with another isn’t available. We integrated the IBM Aglets API with a Java library in order to access a smart card and use digital signature technology in agent programming. New Java API have been developed in order to sign code and data of an agent application. The programmer can choose to dispatch its agents using digital signature to authenticate their classes, when it needs to dispatch the code, and the agent’s state. The hosting server can check the integrity of received bytes and authenticate them. The server can be initialized to store the received signed code in order to grant itself from sender’s repudiation.

8 Integration of MAGDA with an OpenMP Compiler

The Mobile Agents and high-level shared memory programming paradigms can be coupled in order to express a hierarchical (two-level) parallelism: an external distributed memory level, and an internal shared memory one. The interacting agents’ execution model perform the external distributed memory parallelism; internal shared memory parallelism can be achieved within each agent’s execution, and can be expressed through OpenMP directives.

We have obtained such a coupling through the integration of MAGDA with OpenMP compiler technology. In particular we have utilized the JOMP Compiler [6], developed by the EPCC research center of the University of Edinburgh. It is an OpenMP compiler for the Java language, augmented with OpenMP like directives and methods, proposed by the authors. It provides with an implementation of most OpenMP directives for parallel regions, synchronization and mutual exclusion among threads. It provides a runtime library, in the form of a Java class library.

The integrated use of JOMP within the Aglet workbench is straightforward: The JOMP compiler is to be used just as a preprocessor for the Aglet Workbench, for the Java agent classes whose methods contains OMP annotations. More specifically, it is needed to:
• write the agent class containing OMP directives in a file FileName.jomp
• import in it the jomp.runtime library:
• compile the jomp file with the command: Java jump.compiler.Jomp FileName
• compile the file FileName.java obtained in the last step using the Java compiler;
• recall the FileName agent class in the Tahiti console to create the agent.

9 Related work

Many Java based system have been proposed for meta-computing. Some of them implement in Java the features of classical message passing based systems such as MPI and PVM: some relevant examples are JPVM, JavaPVM and ICET [11],[20],[4]. Jpvm is a Java library for explicit message-passing based parallel programming in Java. It offers classical PVM services together with some new extensions such as thread safety and multiple communication. IceT proposal is similar, but it provides the additional functionality of harnessing and merging with remote resources [4] where the user lacks privileges and directly supports the uploading of processes and data upon aggregated resources. JavaPVM is layerd upon the standard distribution of PVM. It uses some of the original PVM code thanks to the capability of Java to call function written in other languages. Some portion of the software can be written in Java, while C and Fortran could be used to write the remainder of the code in order to improve performance and reuse existing code. A similar purpose, with reference to MPI library, is targeted in [15].

Another trend however deals with using mobile code, applets and mobile agents in the job to solve distributed problems. Jevelin [9] distributes the jobs among a massive configuration of applets located on the available hosts. Java architecture provides security features, fault tolerance and load balancing functionalities are implemented while message passing is implemented forwarding the messages to a broker. Charlotte [8] allows any machine on the Web to participate in any ongoing computation. It transmits the program to participating machines, provides a distributed shared memory abstraction at the programming language level and supports load balancing and fault tolerance through two different scheduling strategies. Bayanihan [17] employs Java applets and applications in order to solve a problem made of a pool of tasks. Each user is a volunteer who asks a task, solves it and returns the results, just accessing a web site. The server manages load balancing, communication, collection of results and fault tolerance. Mobile agents technology provide a ready implemented mechanism for the delivery and execution of task code on remote machines. Several mobile agent packages currently provide the basic functionalities together with other features about security, communication, user interface and management. Many parallel programming environments are based on a mobile agent platform which is used such as an engine of the proposed architecture.

Voiajet, AgentTcl, APRIL, Odissey, Grasshoper and Aglets are among the best known mobile agent platforms. Many works about the employment of mobile agents in Agent Based High Performance Distributed Computing can be found in literature [16, 12, 13]. Pacman [10] employs the Aglets workbench and add a software structure in order to provide agents’ coordination and load balancing features. WYA (While You are Away) is based on the NOMADS mobile agent system. It provides dynamic load balancing moving agents toward free and idle workstations. A coordinator distributes roaming computations among the idle workstations. When a workstation become busy the agent come back to the coordinator [18]. In [19] two different load balancing strategy (the first one centralized and the second one is distributed), based on agent’s mobility, are designed and tested on small and large clusters of PCs in order to optimize the performance results.

The main purpose of MAGDA is instead to support parallel programming over distributed platforms through a middleware which implements the needed services of classical parallel programming environment, revisited within the mobile agent based approach. We added some message passing features like collective communications to our platform. The mobility allows to move the computation dynamically on the nodes of a network when a new host grows up or become idle. The redistribution of agents is performed in order to optimize the system utilization and the application throughput and dynamic reconfiguration. The agents’ synchronization and coordination can be performed at the best by means of classical MP mechanism. Collective communication primitives and Open-MP extension are some typical examples. The MAGDA’s purpose is to make easy the utilization of the original sequential or parallel code to the programmer and to hide the difficulties dealing with synchronization, communication and coordination in mobile agent programming, through programming skeleton, execution pattern and high level primitives.

References


[16] C. Muthukrishnan and Suresh T. B. Indian Institute of Technology, Madras, India,” A Multi-Agent Approach to Distributed Computing”, Agent Based High Performance computing, Seattle, Washington, USA May 1, 1999


