8^{th} *International Conference of Modeling and Simulation - MOSIM'10 - May 10-12, 2010 - Hammamet - Tunisia*
"Evaluation and optimization of innovative production systems of goods and services"

ADAPTIVE RUNTIME FOR NUMERICAL CODE

Cyril Dumont

LACL, Paris 12 University
61 avenue du Général De Gaulle
94100 Créteil, France
dumont_cyril@yahoo.fr

Fabrice Mourlin

LACL, Paris 12 University
61 avenue du Général De Gaulle
94100 Créteil, France
fabrice.mourlin@wanadoo.fr

ABSTRACT: *A strategy for the analytical solving of ordinary differential equations and a first implementation of it based on mobile agent community, using jini Numerical algorithms are already designed in many domains such that electromagnetism, fluid theory. Numerical codes were developed from these mathematical analyses with interesting performance. Our work is about our runtime platform to execute existing numerical codes by the use of mobile agents. They act as workers which manage parts of a numerical code on a set of computers.*

KEYWORDS: *mobile agent, space computing, numerical analysis, re-usability.*

1 INTRODUCTION

Numerical analysis naturally finds applications in all fields of engineering and the physical sciences, but today, the life sciences and even the arts have adopted elements of scientific computations. When scientists and engineers need numerical answers to mathematical problems, they turn to computers or more specific computing architecture. Nevertheless, there is a widespread misconception about the process of computation. The role of numbers is essential. During a research process, scientist set numerical measurements which led to physical laws expressed mathematically. So, these formal expressions are a basis for numerical algorithms, but the role of computer is more complex.

In many cases, a mathematical expression cannot be transformed into an automaton or a sequence of operations. Because, it is not always possible to build a precise and fast algorithm, this construction changes to an approximate fast answer that is accurate to ten or twenty digits of precision. For scientific application, such an answer is often as good as exact. Numerical analysis is the study of algorithms for solving problems of continuous mathematics. A lot of subjects are touched such that to solve a system of n linear equations in n unknowns of to find the shortest tour between p cities. But famous algorithms already exist about solver of partial differential equation (DPE) or solver ordinary differential equation (ODE). Various declinations of such algorithms exist depending on the input data types, or on this data size. This means that a second problem arises for engineer: how to use these algorithms on a common platform. Because an ODE algorithm is defined on a specific cluster of computers, it is not usable on a workstation easily. Also an engineer has to manage several configurations due to his current work. We decided to assist engineer to adapt execution of selected numerical code (Hairer & Wanner 1987). Our work is not to create or to change an existing code into a more suitable one, but we want to use numerical codes which are already validated in an adaptive environment.

The adaptation has several facets: one of them is used during execution. If a hundred processors are available, a simulation can use them. But if during this execution, some of them have to be stopped or used by someone else, and then the execution must continue onto a subset of processors. A second facet is about the scheduling of numerical code (Powell 1981). Frequently code is divided into parts which are totally independent; also parallelization is possible if there are enough processors. The adaptation is to treat these parts concurrently or not depending of execution state.

Our solution (Dumont & Mourlin 2007, Dumont & Mourlin 2008) is based on the use of mobile agent and the construction of agent coalition: one per computing case study. Previously, we present results about numerical code written into Java language, because this programming language is compatible with our platform called MCA for Mobile Computing Architecture. Now, we present through this document a more powerful approach where existing numerical code is used in the original programming language, but the scheduling is done through the use of mobile agents. The next section is about the use of mobile agent into a mobile coalition. Next we detail technical ar-

chitecture of MCA platform and precisely the role of Worker agent. Next we explain a case study about polynomial equation solver and our results (Nocedal & Wright 1999). Finally, we highlight key points of our work and the next step of our research approach in the domain of computing technology.

2 MOBILE AGENT AND COALITION

Mobile agents are software abstractions that can migrate across the network (mobile characteristic) representing users (called agents) in various tasks. Mobility has advantages over static alternatives with benefits, such as improved locality of reference, ability to represent disconnected resources, flexibility, and customization.

2.1 State of the art

The term "mobile agent" was introduced by script languages, which supported mobility at the programming language level. Many mobile agent systems followed, most implemented in Java, which already supports mobile code, but also in scripting languages, such as Perl or Python. Mobile agents seem suitable for applications, such as web commerce, system administration and management (especially network management), and information retrieval. However, few systems actually deployed them in an industrial setting.

Research projects already exist about platform based on mobile agent such as D'Agents at Dartmouth University (Gray, Kotz, Cybenko & Rus 2000), or Tacoma mobile-agent system as well as the StormCast distributed application. But all these systems need to develop new codes with specific guidelines. And because mobile agent concept is not normalized, it is not possible to reuse a mobile agent application from one platform to another one. Also, we designed our platform with a first objective to be adaptive. It means an ability to reuse existing code into a runtime context managed by mobile agents (Cabri, Leonardi & Zambonelli 2000).

2.2 Role of mobile agent

Mobile agents can be deployed in distributed systems because mobile agents have a number of key features that allow them to reduce the network load and overcome network problems. They can encapsulate computing protocols, and they can work remotely, even asynchronously and disconnected from its group called coalition.

The idea of mobile code is easy to understand, but it presents all sorts of interesting technical challenges. Distributed computations are somewhat more complex. The key concept of mobile agent is about its migration process.

First, migration is used by an agent to access to an available resource. This is particularly useful when there are a lot of processors and some of them are available for a computation and some others become free during first computing step. Thus, it is possible to take up these new resources. In our context, it means that mobile agents help to the management of a computing case study (Lal & Pandey 1999) (Tschudin 1997).

Secondly, migration needs to manage lookup services where agents are recorded. When a resource becomes free, its access is saved into a lookup service. Also, when a failure occurs, the states of all the lookup services are an indicator to retrieve a stable state of a computation. For numerical codes, it means that mobile agents allow restarting an interrupted execution.

Third, migration is also a key feature for dynamic deployment by launching mobile code from a central place. Then, they migrate and reside at different computers, rather than installing copies of the software at all servers individually. This involves that all numerical code are saved on to some computers, maybe one, and their parts are exported to free computers. Of course, if a new version of the agent is available, computers are informed by remote events. The new version agent migrates all computers. All agents are then synchronized.

This migration concept relate well with several technologies such as CORBA and Jini (Java Intelligent Network Interface). While both technologies are quite similar, Jini has some advantages : pass-by-value and pass-by-reference (CORBA only does pass-by-reference), dynamic loading interface, Leasing mechanism, multicast lookup request, variety of protocols (CORBA only uses IIOP) ... . For these points, the useful services available in that framework and our experience in previous research activities, we decided to use Jini framework.

2.3 Mobile agent coalition

When a platform is just based on a set of mobile agents, it means that only one computing case is considered on that platform. A structure has to be created like a group or coalition. In our context a collation is a group of mobile agent with a common objective: a numerical case study with a specific code and selected input data.

Through a more technical approach, a collation is a space where interaction can be done without perturbation. Also, we develop this concept to manage several computing case studies on to the same set of computers. Agent communication for the mobile agent's

coalition means, at least most of the time, the exchange of objects or object references. This can be message or more partial results. At a higher level, a coalition can have its own communication strategy and specific rules for the data persistence (da Silva & da Silva 1997).

In previous section, we explained about the role of the agent migration. Now, we can extend this concept with migration strategy for a coalition. On a grid of processors, this corresponds to a wave where all agents move by application of the same migration law. This kind of rule can be based on the lookup of a kind of resource such as a specific solver or a particular size of memory or more powerful a load balancing decision from the master of the coalition.

3 AN ARCHITECTURE ADAPTED TO NUMERICAL SIMULATION

In this part, without going into details as we did previously (Dumont & Mourlin 2007, Dumont & Mourlin 2008), we will describe our architecture. We can highlight two features of it: the adaptability and the mobility. Java has become necessary in terms of programming language; effectively, it allows any OS with a Java virtual machine to execute a Java byte code. All components (*ComputingMaster*, *ComputingWorker*, *ComputeAgent*, *MCASpace*...) of our architecture are developed in Java. We will see their utility as a first step.

Java allows us to accept any type of OS or architecture (with few exceptions). This means that components of our architecture (especially the *ComputingWorker*) can easily adapt to different contexts. But, in addition to its execution context, a component must also adapt to the calculation case in which it wishes to participate. The whole calculation case (its tasks, data, properties and of course its algorithm) must be mobile and ready to be hosted by a *ComputingWorker*. In a second part we will see why MCA is also mobile agents architecture.

3.1 A Space Based Architecture

To use mobile agents in Java, we decided to use the Jini framework (cf Section 2.2). Among the many services available in that framework, 2 are particularly useful for our architecture: its Lookup service (Discovery and Join) and its JavaSpace implementation (Outrigger API).

Our architecture is based on a space: the *MCASpace*. As we have seen in section 2.3, we can execute several computation cases simultaneously. One computation case becomes an agent coalition in the space. As a shared memory, a coalition contains different types of entries (*Task*, *MCAProperties*, and *Datahandler*).

At the beginning of a case will involve the *ComputingMaster*: it initializes the case. 2 types of computation case are possible:

- The case requires input data, the *ComputingWorker* cuts data into several files and it writes matching *DataHandler*: ie a link (FTP ,SFTP, HTTP, Local...) to a file.

- All tasks for the computation case are not all dependent on them. They are defined before starting the case. The *ComputingMaster* plays the role of a scheduler (Otherwise it is the task itself that write the following tasks in the *MCASpace*).

In all cases, the *ComputingMaster* writes the first task to execute and then signals the end of the computation case to the coalition of agents involved in the latter.

Finally, the *ComputingWorker* is the component that executes the algorithm of calculation (using the *ComputeAgent* we will see in the next section). The number of *ComputingWorker* may vary during the execution of the case.

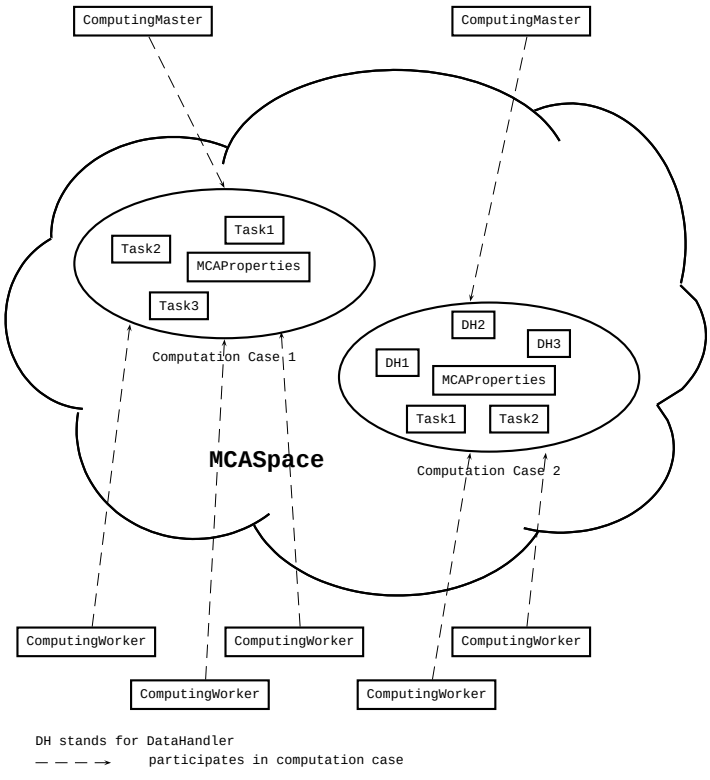

Figure 1: Components of MCA

3.2 A Mobile Agents Based Architecture

We just see the static components of our architecture. But its main feature is the adaptability: an incoming *ComputingWorker* should not disturb the execution of a computation case. For this we use the mobility of different ways.

Firstly, we saw that the *MCASpace* contains several entries. By definition these entries are mobile. The *ComputingWorker* can take data, tasks and properties of the computation case without being configured.

Secondly, all *ComputingWorkers* have the Jini Lookup service enable and listen the arrival of a particular agent: the *TaskNotifierAgent*. This agent is launched by the *MCASpace* upon arrival of a new task. It travels from lookup to lookup (of *ComputingWorkers*) to find a free *ComputingWorker*. Upon its arrival on a lookup, the agent checks the state of the *ComputingWorker*: if WAIT_FOR_COMPUTE then the agent takes the *Task* on the *MCASpace* and launches its execution on the *ComputingWorker*, else the agent continues its roadmap.

Third, a *ComputingWorker* arriving in a computation case does not have the algorithm. Each task is associated with a part of this algorithm. This part is held by a *ComputeAgent*. The *ComputingWorker* retrieves this agent and executes it locally.

4 EXECUTION OF NATIVE CODE

In this chapter we focus specifically on the execution of a task of a computation case (a *Task* entry in our architecture). Then the couple *ComputingWorker/ComputeAgent* is highlighted. But why come back to this couple? Its ability to run existing code. In the previous chapter, we described our architecture as full-Java architecture. But this architecture gets ready to receive native code.

Let's go back when the *ComputingWorker* gets a Task on the state WAIT_FOR_COMPUTE. At this step, 2 cases are possible:

- The simplest case: the code of the computation case is developed in Java code. The *ComputeAgent*, necessary for the executions of the task, is a class that implements the Java interface *ComputeAgentInterface* and redefines the method execute. Then the code is completely written in Java language and can use multiple Java API available. This case is simple but certainly the least used.
- The second case: this time, the code of the computation case is already existing and developed in another language (C/C++). It is here that adaptive runtime for numerical code makes sense.

It is on this second case that we will linger throughout this chapter. It implements several mechanisms that are explained below. But before we begin detailing the implementation of a "native" agent, we will make a brief presentation of LLVM.

4.1 LLVM : Another virtual machine

The Low Level Virtual Machine, generally known as LLVM, is a compiler infrastructure, written in C++, which is designed for compile-time, link-time, runtime, and "idle-time" optimization of programs written in arbitrary programming languages. Originally intended to replace the existing back-end in GCC with a more modern substrate, the success of LLVM has since spawned a wide variety of new front-ends intended to work with it and replace larger portions of the GCC stack.

How to use LLVM architecture in MCA? 2 features are particularly useful: A Just-In-Time (JIT) code generation system and APIs tools to simplify rapid development of LLVM components (Begeman 2008).

4.2 A native agent

We saw in the previous chapter the configuration of a *ComputeAgent*. A native agent has the same properties except that we must to specify the file containing to execute (a LLVM bitcode file).

But it being a subclass of *ComputeNatifAgent* that the *ComputeAgent* is actually a "native" agent. This class differs from the interface *ComputeAgentInterface* by its ability to execute native code. For this, the agent will use the mechanism proposed by *JNI* (Java Native Interface).

JNI is a programming framework that allows Java code running in a Java Virtual Machine (JVM) to call and to be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages, such as C, C++ and assembly. In our case, we use the ability to call a dynamic library developed in C++.

Adaptability is a key feature of our MCA architecture; it seems obvious that the same code will run on several types of OS or architecture. Therefore, the use of the virtual machine provided by LLVM is essential to every *ComputingWorker*.

4.3 A runtime strategy

But how to give the possibility to *ComputingWorker* use this virtual machine? In linking the 2 previously

mentioned concepts (JNI and LLVM) that we will be able to run native code in our Java architecture.

As we said in our presentation of LLVM, we used the C++ API provided (which allows among others to communicate with the virtual machine). We have developed a dynamic library called *libMCA.so*. This library provides 2 functions. Here is the definition:

```
void load(const char* file);
const char* execute(const char* function,
        const char* parameters[]);
```

The first method loads a LLVM bitcode file and the second executes a function (defines in the file) with parameters if necessary, it may return a result.

In definitive, JNI does not directly used to execute native code. But it is our library, loaded using JNI, which will execute the code contained in a LLVM bitcode file. This file will then be able to interpret any machine installed with LLVM. But how comes this file there until *ComputingWorker* ?

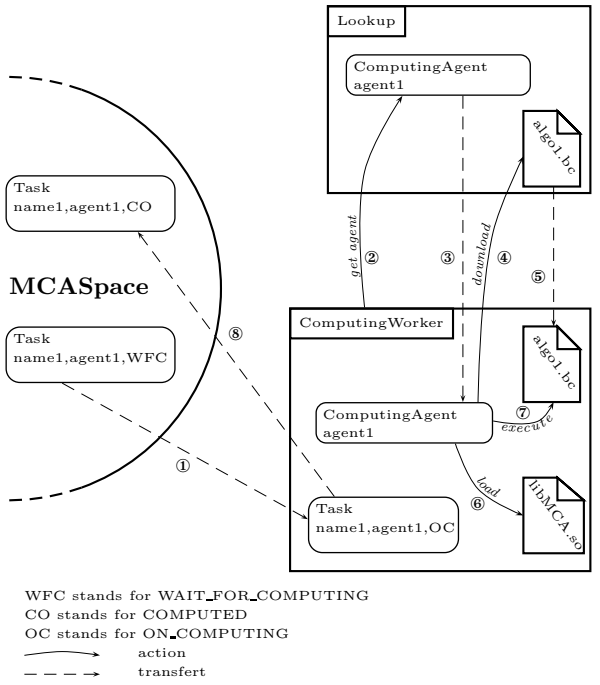

Figure 2: Execution of a native task

The LLVM bitcode file is strongly linked with the agent (We saw it during the configuration of the agent indicates that the file is associated). Whether the agent is native or not, it remains less mobile and *ComputingWorker* will be able to recover it. The *ComputeAgent* is now on the *ComputingWorker* (until now, nothing to change compared to a traditional agent).

Here are the specific steps to the execution of a native agent:

1. If this is the first execution of this *ComputeAgent*, it itself will retrieve the llvm bitcode file using the *DataHandler* (in the same manner used to retrieve data needed for computations cases).
2. The *ComputeAgent* (running in the Java Virtual Machine of the *ComputingWorker*) loads the MCA library with JNI and with it, loads the llvm bitcode file and execute the function necessary for the execution of the task.

The task was performed normally and the normal process continues.

5 CASE STUDY: CUBIC EQUATION SOLVER

Thermodynamics is based on rigorous mathematics; the problems it solves require a considerable degree of ingenuity and creativity, particularly about equation of state. The use of cubic equations of state is not new for process engineers, especially those in the natural gas and petroleum refining industries. Nevertheless, the convenience of having a flexible model for the representation of strong liquid phase using an equation of state is not always recognized by users of activity coefficient models (Satyro 2000). For example, an equation of state with good mixing rules can provide vapor/liquid equilibrium information, but also volumetric and calorimetric properties, as well as natural handling of supercritical conditions. These characteristics are more and more important as synthesis and separation processes are modeled together for economic reasons, and a uniform model for high and low pressure conditions is of evident value.

5.1 Mathematical approach

In mathematics, a cubic function is a function of the following form where a coefficient is nonzero; or in other words, a polynomial of degree three.

$$f(x) = ax^3 + bx^2 + cx + d \tag{1}$$

Every cubic equation with real coefficients has at least one solution x among the real numbers; this is a consequence of the intermediate value theorem. We can distinguish several possible cases using the discriminant Δ:

$$\Delta = -4b^3d + b^2c^2 - 4ac^3 + 18abcd - 27a^2d^2 \tag{2}$$

This approach involves complex computations, and their cost is high when the number of cubic equations

is large. A faster solution is built from Cardano's method (Dunham 1990). This method is divided into a sequence of steps which starts with a substitution: $x = t - a/3$ eliminates the quadratic term, giving the so-called depressed cubic. The simplified equation is:

$$0 = t^3 + pt + q \; where \; p = b - \frac{a^2}{3} \; and \; q = c + \frac{2a^3 - 9ab}{27} \quad (3)$$

The second step is another transformation based on the introduction of two variables u and v linked by the condition $u + v = t$, and substitute this in the depressed cubic giving:

$$0 = u^3 + v^3 + (3uv + p)(u = v) + q \quad (4)$$

At this point Cardano imposed a second condition for the variables u and v: $3uv + p = 0$ which gives a more simplified equation:

$$0 = u^6 + qu^3 - \frac{p^3}{27} \quad (5)$$

This can be seen as a quadratic equation in u^3. The last step is the solving of this equation, we find that

$$u = \sqrt[3]{-\frac{q}{2} \pm \sqrt{\frac{q^2}{4} + \frac{p^3}{27}}} \quad (6)$$

The expression above for u can generate up to three values (there are three cubic roots related by a factor which is one of the two complex cubic roots of one, and two square roots of any sign ; but these 6 expressions can generate only 3 pairs). This also applies to the final solutions for x.

5.2 MCA implementation

Our computing platform allows users to use numerical codes written in several languages. Also, we selected a C++ code (Kaw 2009) which solves cubic equations with algorithm based on section 5.1 approach. This code is divided into five steps: two transformations, a computation, and two reverse transformations. At the end the x variable is evaluated.

The activity of that code is easily scheduled. In a sequence environment, it gives results from cubic equation. But our need is to solve equations of state belonging in a set which grows continuously due to thermo dynamical experiments. So, the code execution is not so fast and the cardinality of the input set doesn't reduce at all. Also we decided to use this code in our adaptive environment. A space is created to receive a mobile coalition. The space contains parts of the code called task, the workers which execute these tasks and the set of cubic equations. An equation is described with four coefficients. We choose to use 16 computers in a first experiment and we observe reduction of the input set and a stop of the activity when the input data set became empty.

Moreover, when new equations were added into the input data set, the computation restarts and a new deployment of mobile agents occurs. This phenomenon highlights properties of our approach: autonomous, dynamic and adaptive. The output data set contains results for each input equations and now we try to solve cubic equation systems, this introduces correlation between solutions.

6 CONCLUSION

To sum up, we presented quickly our computing environment and a new case study, a cubic equation solver. This case highlights new specific features to test: costly execution, written in a programming language distinct from our platform. Our research has instead focused on reuse and not on performance. We obtained results which confirm this research approach. Our initial desire was to use existing numerical code and our architecture is able to provide mobility to a code which is not mobile initially. Because our architecture is open, we are confident that next step will be an execution of Fortran code (very popular in numerical codes field) into a mobile agent environment.

REFERENCE

Begeman, N. (2008). Building an efficient jit, *2008 LLVM Developers' Meeting*, Apple Campus.

Cabri, G., Leonardi, L. & Zambonelli, F. (2000). Mars: a programmable coordination architecture for mobile agents, *IEEE Internet Computing* **4**(4): 26–35.

da Silva, M. M. & da Silva, A. R. (1997). Insisting on persistent mobile agent systems, *First International Workshop on Mobile Agents*, Vol. 1219, Springer-Verlag, Berlin, Germany, pp. 174–185.

Dumont, C. & Mourlin, F. (2007). A mobile computing architecture for numerical simulation, *Proc. International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies UBICOMM '07*, IEEE Computer Society, pp. 68–74.

Dumont, C. & Mourlin, F. (2008). Space based architecture for numerical solving, *Proc. International Conference on Computational Intelligence for Modelling Control & Automation*, IEEE Computer Society, pp. 309–314.

Dunham, W. (1990). *Journey through Genius: The Great Theorems of Mathematics*, Wiley, chapter 6 : Cardano and the Solution of the Cubic, pp. 133–154.

Gray, R. S., Kotz, D., Cybenko, G. & Rus, D. (2000). Mobile agents: Motivations and state-of-the-art systems, *Technical Report TR-2000-365*, Department of Computer Science, Dartmouth College.

Hairer, E. & Wanner, G. (1987). *Solving Ordinary Differential Equations*, Vol. I and II, 2nd revised edition edn, Springer, Berlin.

Kaw, A. (2009). Exact solution to cubic equations, Textbook notes on finding the exact solution to a cubic equation. General Engineering.

Lal, M. & Pandey, R. (1999). CPU resource control for mobile programs, *First International Symposium on Agent Systems and Applications (ASA'99)/Third International Symposium on Mobile Agents (MA'99)*, IEEE Computer Society, Palm Springs, CA, USA.

Nocedal, J. & Wright, S. J. (1999). *Numerical Optimization*, Springer.

Powell, M. J. D. (1981). *Approximation Theory and Methods*, Cambridge University Press.

Satyro, M. A. (2000). Modeling vapor–liquid eguilibria: Cubic equations of state and their mixing rules, *Chemical Engineering Progress* .

Tschudin, C. F. (1997). Open resource allocation for mobile code, *First International Workshop on Mobile Agents*, Vol. 1219, Springer-Verlag, Berlin, Germany, pp. 186–197.