RESOURCE SHARING AND LOAD BALANCING BASED ON AGENT MOBILITY

Gilles Klein
LAMSADE - University of Paris IX
Place du Maréchal de Lattre de Tassigny, 75016, Paris

Alexandru Suna, Amal El Fallah-Seghrouchni
LIP6 - University of Paris VI
8, Rue du Capitaine Scott, 75015, Paris

Keywords: Mobile Agents, Agents for Internet Computing, Load Balancing and Sharing, Agent-Oriented Programming.

Abstract: From the recent improvements in network and peer-to-peer technologies and the ever-growing needs for computer might, new ways of sharing resources between users have emerged. These methods are very diverse, from SETI@HOME which is a way to share the load of analysing the data from space in order to find traces of extraterrestrial life, to NAPSTER and its successors, and to Real-time video-games. However, these technologies allow only centralised calculus-sharing, even if they already offer "peer-to-peer" sharing of data. We present in this paper a method based on Multiagent systems which allow load-sharing between distant users.

1 INTRODUCTION

As the power of computer processors keeps growing, it appears that whatever the power of the computers is, it is never enough, as computer software is always more demanding. Even if it is now common to own a computer, the most powerful ones are still out of reach of most of the users. As distributed applications become very common (multiplayer games, for example), network-based sharing technologies are still limited to very specific applications (like peer-to-peer sharing systems). Even the great societies could vastly profit from it, simply because the computer resources of multinational groups are always underused. As a matter of fact, as these resources are distributed all around the world, at every single moment, there are computers unused, simply because their user is not working at that time. These computers could be used to lower the load of the ones placed at "the antipodes". These points are what drives research into the domains associated with grid programming (Foster and Kesselman, 1999) and peer-to-peer systems, from SETI@HOME (Seti) to Napster but also to works like (Vercouter, 2002) on open systems working without middleware agents.
To permit sharing of resources (not only between employees of a single group, but also between anonymous users), certain problems must be overcome.
To be efficient, a resource sharing application must allow users to execute code on the shared computers. It induces a clear problem: The code that will be executed must be present on the shared computer; it makes real sharing very difficult as it is difficult to know a priori what application will be shared and so these applications cannot be installed on every shared computer beforehand.
The sharing system must be flexible as it must neither disturb the users nor be hindered if a computer is turned off.
The system will not be used if it is needed of everyone to reboot their distributed applications every time one of the shared computers suffers a problem (an event which can happen relatively often if the computers are personal systems). It becomes all truer if the sharing system is distributed on a large scale (e.g. Internet).
It must also be considered that as the system would be then very wide, it would not be possible to make the hypothesis that the communications are timeless, so local information of a site concerning another site will always be obsolete (even if only by few seconds). In fact, trying to renew the local information often might be so costly in terms of communications, that the whole system would be diminished. So the load sharing must be done based on obsolete information.
The system must also guaranty the safety of the users. In fact, this kind of systems implies that foreign code will run on the shared computers, which can be a risk for every user.
Finally, for the sharing to be efficient, it is necessary that the applications which are going to be distributed

Klein G., Suna A. and El Fallah-Seghrouchni A. (2004).
RESOURCE SHARING AND LOAD BALANCING BASED ON AGENT MOBILITY.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 350-355
DOI: 10.5220/0002601303500355

on it will use it correctly.
It is not reasonable to believe that it might be possible to distribute very heavy applications efficiently while all the distribution is managed by a "super-OS". Highly distributed systems are different from shared-memory multiprocessors computers because the communications between the computers can be very expensive when compared to the internal communications of a computer. But the system itself has no way to determine the cost of the communications, if the application is not built specifically so that it includes a description of its processes' communications.

2 RELATED WORK

The resource sharing between different computers is a goal of several distributed computing approaches, from Grid-Computing to Cluster-computing or peer-to-peer sharing systems. Our model has common elements with each one of these.
In many aspects, we follow the same basic principles as the ones of the peer-to-peer systems of the last generations (Kazaa, Overnet (Bhagwan, et al., 2003) or Chord (Stoica, et al., 2001)). Those systems follow certain hypotheses that should be followed in the case of all resource sharing systems:
- Since the problems of Napster, every one of these systems must be fully distributed, so that no central server could be attacked (physically or legally).
- They must be generic; every kind of file can be shared using these systems. It was not the case for Napster, which offered only MP3s sharing, but since Gnutella, the sharing systems are not restrained by the type of the shared files.
- They are anonymous. In fact, they are not anonymous, the requests are anonymous but the answers are not (it is so to let two computers communicate directly). However, the principle is very good, as anonymity offers the greatest protection against specified attacks.
- The mass of communication is minimized. To reduce the communication load, several methods are applied, from "super-nodes" architectures (e.g. Kazaa) to indexing methods (e.g. Overnet).
However, there are evident differences; first, it is far easier to share files than generic computer resources. File sharing does not ask for neither load-balancing system nor redundancy and other safety systems. It is why, while in principle, sharing computer resources within the characteristic hypotheses of the peer-to-peer file sharing systems, we should also consider the works in grid computing and cluster computing as they both treat the problem of distribution computational load on several computers.
The research in the clusters domain encountered a serious problem while treating the case of heterogeneous computers. Several system exist that accept heterogeneous computers, for example PVM (Geist, et al., 1994), P4 (Dongarra, et al., 1993) or LINDA (Carriero and Gelernter, 1989). One of the most interesting solutions concerning cluster computing has been the introduction of distributed shared memory like in LINDA. The functionalities of the clustered systems have also been applied to larger scale systems (e.g. using Globus (Foster and Kesselman)). However, distributing a shared application (e.g. SETI@Home project (Seti)) is easier than sharing computational power for applications unknown. The only solution to really share resources is to use mobile code, to have the processes migrate from a site to another until they find a good one.
Most of the grid programming systems (Globus, Legion (White), MPICH (MPICH-G2)) offer the functionalities necessary to implement every kind of distributed application. However, these systems are rather platform for building distributed application than sharing platform. Most do not implement mobility facilities and so cannot be used in the frame of our hypotheses.

3 MOTIVATING OUR CHOICES

3.1 An architecture for mobility ...

Multiagent systems appeared as the most pertinent basis to build our sharing system as it helps to solve many of the problems cited before. They offer a great flexibility to the whole system and are naturally adapted to distributed environments. Nevertheless, as we showed in (Klein et al., 2002), in order to be plainly able to profit from the diverse advantages of distributed systems, they must be conceived so that they take into account these distribution problems. Using mobile agents gives us a solution to most of the problems described before; indeed, mobile code offers the lone alternative to the presence of a copy of every single distributed program on every shared computer and the agents are not only very adapted to the distributed systems, but they also carry a very high level of encapsulation of the code which can help to improve the security of the system and a certain autonomy of the processes distributed across the network which can let the system be very adaptive to the evolution of the situation.

3.2 ... and resource sharing

As we showed before, it would be interesting to be able to share resources from different users' computers even if they are distant. To be able to do that,

we have to build a sharing ”platform”. Three different points must be taken into account: the security and the managing of every single computer participating to the sharing network, the building of every of the tasks distributed on the different computers and finally, the state of the system as a whole.
As sites can be owned by different people and as the users can have divergent goals and as the number of shared sites is not limited, it is not realistic to believe that we can build a sharing system following a real hierarchic architecture. There can be neither centralized management nor even domain-based management. So we considered an agent-based architecture composed of three types of agents.
The **Task** agents encapsulate the distributed processes of the users. They verify if the site where they run is satisfying enough, and if it is not, they look for a better one following the pieces of information they possess and the information the Manager of the site can give them. Then they migrate to a better site.
The **Manager** agents manage their sites and they are not mobile. They guaranty that the local running Tasks agents follow the rules, that the security and the stability of their site are good enough and that the local information about the system as a whole is not too obsolete.
The **Sniffer** agents travel across the network, migrating between shared sites to find good sites for the applications of their owners. They also try to renew the information of every site they go through by making propositions of changes about the local knowledge to the manager of the site. So they make the system as a whole efficient as these pieces of information are used by every Task agent. Finally, their contents can be analyzed when they come back to their home site.

4 HOW CAN AGENTS CLASSIFY THE SITES

One of the first problems we met while conceiving a sharing system where agents choose their destinations by themselves can be found in the fact that it is necessary to give the agents a tool to make that choice. So both a computable representation of the computers and a method of classification fast enough to not consume too much resources (the agents must spend as few resources as possible or the distribution will not carry any gain).

4.1 How agents represent the sites

One of the first characteristics of our system of representation is that it must let agents take into account the temporality of the information. Indeed, in order to not overload the communications, we must consider that the knowledge of a site about another one will not be refreshed very often (the Sniffers will refresh them only if they can in their present state). It is also necessary that this knowledge is simple enough to be computed in a reasonable time, still complete enough to offer good results.
We must separate the data about the sites in two categories, the dynamic and the static information. The static data is characteristic to the computer (like its microprocessor or OS) and it can evolve but very rarely; on the other hand, the dynamic information represent the temporal state of the computer at a precise time. The dynamic information will be the one that should be refreshed from time to time.

4.1.1 Static Data

The static characteristics that we are taking into account in our application are: the computation power, memory’s and disks’ characteristics, allowances of the hosted Task agents, network connection characteristics. For our tests, we obtained the first two points with PerformanceTest edited by PassMark Software, which gave us numerical marks for each one of these points on PCs with Windows operating system (OS). It is possible to obtain similar things with the system tools when running Linux, but to simplify this paper we will only consider Windows-based sites. The permissions are fixed by the user and represented in a binary table, same as the connection nature. The average distance is a numerical measure (of time) and the quality of the connection is a distribution curve.

4.1.2 Dynamic Data

The dynamic data are divers curves describing the evolution of the system that are refreshed locally, like the load of the processor, of the memory, the instant distance to a reference site, and the instant quality of the local connection. There are other kinds of data that are not taken into account yet, like the specific services accessible on this site.
For our simulation, we used a simplified version of the description of the sites; it only includes the basic characteristics of the computer.

4.2 Classifying the sites

To classify the computers is a necessity. Indeed, the agents must not only represent the sites but also use these representations to make decisions about where to and when migrate. They must be able to select a site and decide that the services of their present site are not sufficient. So they must be able to compare and classify the sites, satisfying the following rules: classifying a computer must use little resources; the classifying functions must be simple and light; the

classification obtained must be systematic and there must be no ambiguity.
To obtain all these characteristics, we choose Galois's lattices (Ore, 1944) to represent the preferences as they can be applied even to abstract data (e.g. curves, histograms, fuzzy sets) (Diday and Emilion, 2003).
The Galois lattices can be defined as follows: be two lattices $(E, \geq, \cup, \cap)$ and $(F, \geq, \cup, \cap)$ where $\geq, \cup$ and $\cap$ are respectively the order relationship, the *supremum* and the *infimum*, a Galois correspondence is a couple (f, g) so that:
$f : E \rightarrow F$ and $g : F \rightarrow E$ are decreasing mappings $h = f \circ g : E \rightarrow E$ and $k = g \circ f : F \rightarrow F$ are both extensive (i.e. so that $\forall x \in E, h(x) \geq x$, resp. $\forall y \in F, k(y) \geq y$).
In our situation, F is the description lattice (which can be seen as a partial classification of the abstract sites and that is presented in the figure 1) and E is the sets lattice, sets that contains the real sites associated to the descriptions. We combine these descriptions with values representing our preferences.

The lattices are sets of sets of computers (with include an order relationship), associated with their descriptions. So, the user defines the preferences of his agents through the use of the lattices of descriptions of computers. We can guaranty that a machine can find its unique place in the lattice after at most *n* comparisons (with *n* being the length of the longest branch of the lattice). It will always be possible to classify a new site and to compare it to older ones. Yet, we must recall that a lattice is a partial order, so certain computers will not be comparable. For this reason, we also ask the user to give qualities to each of the categories defined by the descriptions.
Inside a category, it will choose a computer from the most restrictive node possible. This solution guaranties that the best solution will be selected, but it let some flexibility to the whole; the agents will, for example, be able to choose their destination by taking into account the moves of their "site mates" and go to a second choice if everyone goes to the first one. As the representation and the classifying tool are relatively light, they will be easy to carry during the migrations. Finally, this system is simple enough to open the potentiality of creating new lattices for new applications.

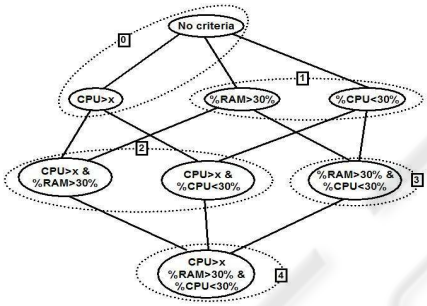

Figure 1: Galois Lattice

5 THE TEST PLATFORM

5.1 CLAIM and SYMPA

For our implementation, we used the CLAIM language and the SyMPA platform (El Fallah and Suna, 2003) because they provide every mechanism for the design of intelligent, mobile agents, and for the visualization of agents' execution and migration.
CLAIM (Computational Language for Autonomous, Intelligent and Mobile agents) is a declarative language that combines elements from the agent oriented programming languages for representing agents' intelligence and communication with elements from the concurrent languages (e.g. the ambient calculus (Cardelli and Gordon, 1998)) for representing agents' mobility. An agent in CLAIM is an autonomous, intelligent and mobile entity that has a list of local processes concurrently executed and a list of sub-agents. In addition, an agent has mental components such as knowledge, capabilities and goals, that allow a forward (reactive behavior) or a backward reasoning (goal driven behavior). An agent can create new agents, can call methods implemented in other programming languages (e.g. Java methods, in this version of CLAIM), can communicate with other agents (one can define using the language's primitives a unicast, a multicast or a broadcast communication) and can migrate using the mobility primitives inspired from the ambient calculus.

SyMPA (French: Système Multi-Plateforme

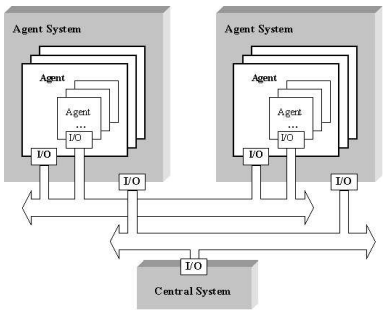

Figure 2: SyMPA's Architecture

d'Agents) is a MASIF (Milojicic, et al., 1998) compliant platform, implemented in Java (Java), consisting in a set of connected computers, that supports CLAIM agents and offers all the mechanisms needed for the design and the secure execution of a distributed

MAS. SyMPA's architecture (figure 2) presents three levels:
The Central System provides services for agents' and agents systems' management and localization.
An Agent System is deployed on each connected computer at the SyMPA platform. It provides high level mechanisms, such as a graphical interface for defining CLAIM agents and classes, an interpret for verifying the definitions' syntax and interfaces for the running agents, low level mechanisms, for agents' deployment, communication, migration and management, fault tolerance and security mechanisms.
An agent in SyMPA is a entity defined using CLAIM. It is uniquely identified. Each agent has an interface for visualizing the behavior, communication and mobility and is concurrently executing the reactive and pro-active behavior.
In order to assure the efficiency and the security of the system, communication protocols were proposed, doubled by cryptographic mechanisms and resources access policies.

5.2 The classes of agents

Two very important characteristics of CLAIM are the generality and the expressiveness. One can represent agents' reasoning, communication and mobility. A Web information research example (El Fallah and Suna, 2003) and an electronic commerce example (El Fallah and Suna, 2003) were already successfully programmed in CLAIM. So we choose CLAIM for our load balancing application.
As we've seen in the application's description, there are three classes of agents.

The **Manager** agents are static agents situated on each of the computers involved in the application. They have a representation of the characteristics of the other sites under the form of Galois lattices. They have capabilities for creating Sniffers, for answering at the Sniffer and Task agents' requests concerning the other computers' characteristics, for updating their knowledge when a Sniffer gives them new information. In plus, when a Manager observes the overcharge of the local resources, it can demand the departure of the local Task agents.

The **Sniffer** agents are created by the Managers and they migrate to the other computers where they update their knowledge about the local computer characteristics, and the knowledge of the local Manager about the other computers.

The **Task** agents are executing several tasks. In order to do this, a Task agent migrates to the best computer known in its Galois lattice, received from the Manager, and starts to execute the task. When it detects that the local resources aren't sufficient anymore, or when it receives a departure demand from the local Manager, it migrates to the new best computer and resumes its task. At the end, it returns to the initial computer and terminates the execution.

5.3 The Practical Tests

It wasn't possible to test our model in a real situation, on the internet, mainly because of the security restrictions concerning the mobile agents. So we were forced to perform the tests at a reduced scale. We used a simplified version of our programs and we installed and deployed them on three different computers from the computational power point of view, connected on a local network (of the LIP6, University of Paris 6), with the following characteristics: a Pentium III-1 GHz, with 1 GBytes RAM and Windows Server 2002 OS, a Pentium IV-2,6 GHz, 512 MBytes RAM, Windows XP and a Pentium III 1,5 GHz, 256 MBytes RAM, Windows XP.
The results aren't entirely representative because they were obtained in a simplified environment. Meanwhile, they give an approximation of the different characteristics, if the system would have been implemented in a real situation.

5.4 Results

The tests were performed in several steps. First, we started four Task agents on each machine, in a static manner (by forbidding the migration). The four Task finish after 20 minutes on the slowest computer, comparing with 10 minutes on the two faster ones. Next, we performed the same test, without started any task on the second computer and allowing the migration. In this way, 2/5 of the system total computational

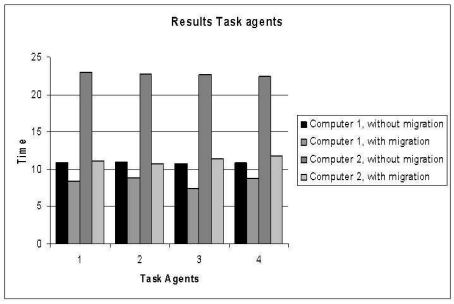

Figure 3: Experimental results

power wasn't utilized. All the agents from the third machine migrated to the first machine (the best one, that was free at the last passage of the Sniffers) which become overcharged, but some of the agents (that overcharged the maximum number of agents for a optimal run) migrated to the second computer, that was free. A centralized distribution of charge would have avoided this useless migration from 3 to 1, but in our

case we couldn't hope for a better execution, because the agents worked with obsolete data. This useless migration took between 20 and 40 seconds for the three involved agents. In spite of this, the earning of time was important (see Figure 3).
We have to note that during the tests, all the agents having the same preference description Galois lattice migrate to the same destination. It is very probable that the situation would be the same if they would have lattices with little differences. This "rabble" migration is not very efficient and the destination computer will be rapidly overcharged. It would be preferable in the future to have a co-operation between the agents in order not to migrate to the same site, because this behaviour affect all the agents in the system. In plus, we'll consider in the future the introduction of authorisation and reservation mechanisms that could avoid the useless migrations of the agents.

6 CONCLUSION AND FUTURE WORK

The performed tests cannot be considered complete for several reasons. First, using SyMPA, the agents' execution is necessarily slowed down in order to allow the users to follow the agents' behavior, communication and migration. Another limit of these tests is their reduced scale and the fact that they don't take into account all the computers selection criterions from our list. It would be necessary to replace the system in a real situation for a complete validation. The system itself leaves some points in suspense. The Galois lattices should be built entirely in a dynamic and automatic manner. It would be also useful to have standard lattices for some types of applications.
Nevertheless, the interest of this work consist in the real experimentation of the agents' distribution, that allowed us to show the viability of the mobility for the load balancing. The computers' classification based on the Galois lattices also seems pertinent. It allows to reasonably choose a computer based on completely dynamic descriptions. Finally, we could prove that the platform efficiently manage the migration and the communication.
There are a lot of things to do in the future concerning this work. In the first phase, we would like to implement our system in an environment closer to its real destination, i.e. on several computers physically distant. We should also propose a coordination mechanism between the Task agents in order to avoid the mass migration to the same destination and a reservation mechanism for allowing a continuous utilization. Using the properties of the Galois lattices, it should be possible to a Task agent to clone a part of itself and to give it the sub-lattice adapted to its needs. It should be also possible to create new lattices in function of the last experiences (maybe using genetic algorithms) and to find a pertinent way to use the data gathered by the Sniffers.

REFERENCES

Bhagwan R., Savage S., Voelke G. M. (2003). Overnet: Understanding Availability. In *Proceedings of the 2nd International Workshop on P2P Systems*.

Cardelli L., Gordon A. (1998). Mobile Ambients. In *Foundations of Software Science and Computational Structures*, Maurice Nivat (Ed.), LNCS, Vol. 1378, Springer, pages 140-155.

Carriero N., Gelernter D. (1989). How to write parallel programs: a guide to the perplexed. In *ACM Computing Surveys*, Vol. 21(3), pages 323-357.

Diday E., Emilion R. (2003). Maximal and Stochastic Galois Lattices. In *Discrete Applied Mathematics*, Vol. 127(2), pages 271-284.

Dongarra J., et al. (1993). Integrated PVM Framework Supports Heterogeneous Network Computing. In *Computers in physics*, Vol. 7(2), pages 166-174.

El Fallah-Seghrouchni A., Suna A. (2003). An Unified Framework for Programming Autonomous, Intelligent and Mobile Agents. In *the proceedings of CEEMAS'03*, LNAI, Vol. 2691, pages 353-362.

El Fallah-Seghrouchni A., Suna A. (2003). CLAIM: A Computational Language for Autonomous, Intelligent and Mobile Agents. In *Proceedings of ProMAS'03*, workshop of AAMAS, Melbourne, Astralia.

Foster I., Kesselman C. (1999). The Grid: Blueprint for a Future Computing Infrastructure. *Morgan, Kaufmann.*

Foster I., Kesselman C. The Globus Project: a statuts report, on-line at: http://www.globus.org

Geist A., et al. (1994). PVM: Parallel Virtual Machine: A Users Guide and Tutorial for Network Parallel Computing. In *MIT Press*.

Java on-line at http://java.sun.com

Klein G., El Fallah-Seghrouchni A., Taillibert P. (2002). HAMAC: An Agent Based Programming Method. In *Proceedings of AAMAS'02*, pages 47-48.

MPICH-G2: A Globus-enabled MPI, on-line at: http://www3.niu.edu/mpi/

Milojicic D., et al. (1998). MASIF, The OMG Mobile Agent System Interoperability Facility. In *Proceedings of Mobile Agents*, LNAI, Vol. 1477, pages 50-67.

Ore O. (1944). Galois Connections. In *Trans. Amer. Math. Soc.*, Vol. 55, pages 494-513.

Seti on-line at http://www.setiathome.ssl.berkeley.edu/index.html

Stoica I., et al. (2001). Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the ACM SIGCOMM '01*, pages 149-160.

Vercouter L. (2002). A fault-tolerant open MAS. In *Proceedings of AAMAS 2002*, ACM, pages 670-671

White B. S., et al. LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications. On-line at: http://legion.virginia.edu/