International Journal of Soft Computing and Engineering (IJSCE)
ISSN: 2231-2307, Volume-2 Issue-6, January 2013

Load Balancing Management by Efficient Controlling Mobiles Agents

Mohamed Bahaj, Abdellatif Soklabi, Ilias Cherti

Abstract- Load balancing is a computer networking methodology which allows the distribution of the workload across multiple computers or computing devices, such as central processing units, disk drives, or other resources, to reach optimal resource utilization, reduce response time, maximize throughput and circumvent overload. The Use of multiple computers with load balancing, instead of a single computer, may increase reliability through redundancy.

Our contribution outlines the adaptation of the Shadow approach used to control mobiles agents for developing a load balancing management algorithm in distributed systems. This approach does not only distribute the loads on the nodes and collect its running result, but it also manages the tasks execution places during all the execution time. Thereby, we get a self-organized load balancing infrastructure.

Keywords— distributed computing, load balancing, mobiles agents.

I. INTRODUCTION

Load balancing in distributed computation is very important to equilibrate the use of resources which are valuable to both applications and system [1] [6] [11] [12] [13] [17]. Both the applications and the system have only a limited amount of them. So it is preferable that applications can utilize resources of different nodes to minimize the running time.

A mobile agent is a computer entity capable of reasoning, running in another remote site, searching and gathering the results, cooperate with other mobile agents, and returning to its home site after completing the attributed tasks [9] [19]. Mobile Agent based applications research are increased and are used to solve many problems of distributed systems [18].

The shadow approach constructs an artificial links between shadow and agents. The shadow agent records all dependent agents' location. Removing the shadow agent implies that all dependent agents are affirmed orphan and finally be terminated. It creates agent proxies that maintain a path from Shadow agent to every dependent agent [5]. This approach can be used to terminate a group of agents even if the location of each single agent is unknown.

In this paper we will underscore a new solution to load balancing using mobile agent control, especially by developing shadow protocol, which is used in orphan detection to allow the balance of the resources access in the distributed systems, and also by adding a receiver agent in charge of sending information about its home node load.

Manuscript received on January, 2013.

Mohamed Bahaj, Department of Mathematics and Computer Science, University Hassan 1st, Fsts, Labo Liten, Settat, Morocco.

Abdellatif Soklabi, Department of Mathematics And Computer Science, University Hassan 1st, Fsts, Labo Liten, Settat, Morocco.

Ilias Cherti, Department of Mathematics and Computer Science, University Hassan 1st, Fsts, Labo Liten, Settat, Morocco.

Besides, our suggestion also involves adding information about the node load from which the request was sent to the time to live sending to shadow agent.

This article is basically divided into many sections, each of which analogically underlines all the aspects of the load balancing based approach. The second section, presents the relationship between load balancing and mobiles agents. The third section, describes how Mobile agent-based applications may run out of resources. The forth section, discusses the Based on Resource Management. The fifth section, concerns itself with a detailed presentation of the Shadow agent model, and the last section deals with the load balancing management by controlling mobiles agents.

II. LOAD BALANCING AND MOBILES AGENTS:

The mobile agent technology can provide various answers to solve the problems of distributing the load in a set of computational entities. Migration of processes was traditionally a solution to this problem, generally under the supervision of a centralized controller. In a modern perspective, multi agents systems can decentralize the distribution of the computational load. In fact, a complex application is divided into autonomous parts, each of which delegated to one or more mobiles agents. Each mobile agent is in charge of searching for the node of the network which offers the most convenient resources, where to execute its own part of code. During the execution, agents can move to other nodes where more computational resources are available, in order to better distribute the load [11].

III.PROBLEM DESCRIPTION

Mobile agent-based applications are planned to allow the number of agent to augment across several nodes, without taking into consideration the agent's mobility during their lifetime. When the agents are activated, they consume a lot of system resources, such as processor power and memory. If an agent is attached to a certain nodes, it often depends on the resources available by them. Hence, if the node runs out of resources, the quality of the service provided by the mobile agent may decrease.

Moreover, extra resources might not work in an optimal way, as it is possible that different nodes are more suitable to run the agent, because of which some nodes may be over loaded, others may be under loaded or some of them may be even inactive. Hence, to avoid this type of state resource management plays an important role and helps in reaching better performance. Therefore, well-placed agents in the network reinforce the optimal use of the resources throughout the system [6] [7].

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

To solve this problem, we need a self-organized load balancing [8] infrastructure which can quickly respond to the various changes in the demand of the various network resources.

IV. STATE OF THE ART IN RESOURCE MANAGEMENT

When tasks enter into a distributed system, the following methods are developed with the goal to disperse them on the various processors [1].

A. Task Assignment Approach

This approach considers each process as a collection of linked tasks which are scheduled to the appropriate nodes so as to get better performance. In this it is implicit that process is already opening into tasks. It conceder also that the power of each processor, processing cost of every task on every node, inter-process communication among tasks, resource requirements of the tasks and available resource at each node are known. Based on above all information an optimal assignment of tasks is found. But reassignment of the tasks is in general not possible in this approach.

B. Load Balancing Approach:

In this approach, all processes submitted by the users are dispersed among the network nodes so as to balance the workload among the nodes by evidently transferring workload from lightly loaded nodes to heavily loaded nodes. Static algorithms employ only information about the average behavior of the system ignoring the current situation of the system. Dynamic algorithms react to the system situation that changes dynamically. Static algorithms are further classified as Deterministic versus Probabilistic. Deterministic algorithms use the collected information about the nodes properties and the processes character to deterministically distribute processes on nodes. Probabilistic algorithms uses information concerning system static attributes such as the number of nodes, processing capability of the nodes and network topology to simplify placement system.

C. Load balancing using mobile agent:

The majority of mobile agent systems are limited to weak mobility [11]. There are a few ones which support the strong mobility, such as JIAC [12] and NOMADS [13]. To implement strong agent mobility, NOMADS is built on the top of a particular Java Virtual Machine that is capable to capture then restore the running state of a Java Thread on different computers. Organic Grid approaches are modeled in a way to simulate complex biological systems organize themselves, by dividing a large computational task into sufficiently small subtasks. Each task will be assigned to one or more mobiles agents, who are then released on the Grid and discover computational resources using autonomous behavior.

JIAC adopts an approach that gives a language to agents' implementation, which is interpreted at Java Virtual Runtime. Strong mobility used in a multi-agent system accelerates the dynamic resource sharing, as provided by diverse Grid systems. However, most Grid computing systems can only decide at which host to start a job and do not transfer load in a dynamic manner.

In the JIAC infrastructure the agents can be inactive or very busy while waiting for new requests, but they use up resources while reacting to a service request. In computing infrastructures, downtimes of service providers should be decreased. JIAC offers relocation transparency which maintains that service requests never get lost. Both the service provisioning activity and the agent movement are resumed on the target nodes. In addition, JIAC does not take for granted that agents are independent of each other.

The problem is that we are not capable to run daemons that report exact and up-to-date sub-cluster node status information to the centralized scheduling software. The only information that we receive from a remote node is the execution time of the last job, and if several jobs were run, we also receive from the remote node the average execution time. The classic expression of static and dynamic scheduling is no longer completely appropriate due to its inability to explain run-time job assignment algorithms, which use minimal information about remote nodes. These algorithms are not strictly static as job placement is performed at run-time. They are also not strictly load balancing algorithms in the sense that a centralized algorithm decides job placement and task until jobs are really ready to be executed [2].

Comet is load balancing algorithm [20] based on the technique of calculating credit by the formula:

$$Ci = -x1\ wi + x2\ hi - x3\ gi$$

Where *wi*: computational load of an agent *ai*
hi: intra-machine communication load of an agent *ai*
gi: inter-machine communication load of an agent *ai*

x1, *x2* and *x3* are application dependent coefficients used to estimate the affinity of the agent to the machine.

To calculate a machine Load the algorithm uses the following formula:

$$Lk = \sum_{M(I)=k} (wi + ui)$$

Where Lk is the charge of the machine Mk calculated by the sum of agents who have located on this machine, knowing that:

ui: the sum of the costs of communication within and between machines.

$$ui = hi + gi$$

$$C = \sum_{M(I)=M(aj)} (ai, aj) + \sum_{M(I) \neq M(aj)} c\ (ai, aj)$$

Where *c (ai, aj)*: number of ticks (unit of time) needed to establish communication between two agents ai and aj
F: factor of degradation of the bandwidth in inter-machines communication.

The agent with the lowest credit will be selected for migration, because this agent spends most of its execution time in communication with remote agents. The location policy is to identify the remote agent that generated the largest flow communication with the agent to migrate. Thus the machine where the agent is selected will be the new destination.

The Limits of Comet is that it does not deal with the following cases:

- If the machine chosen by the location policy is loaded
- If the agent selected by the selection policy does not communicate with any external agent, which machine to choose?

- the algorithm does not specify whether the coefficients *x1*, *x2* and *x3* agent would change after its migration to a new machine or not.

In addition, the algorithm considers that mobile agents system is the only system installed on the machine, or it is not the case in reality, until now there is no operating system based on mobile agent, there is only management software of mobiles agents, so the consumption of resources is shared between applications of mobiles agents and other applications on the operating system.

V. THE SHADOW AGENT MODEL

The shadow agent model basically refers to the concepts of agents and places described in detail in [14], [15] and [5]. Since the shadow is already within the system, the agents don't need to contact either the application or to the computer system containing the application. During intervals called *time to live* the system checks for agents that do not have yet a related shadow agent. Then, they are considered as orphan agents and automatically removed.

In case a new agent is created by another one, the system assigns to the new agent the shadow of the creating-agent. This process causes the agent to load more balancing by creating other agent, whereas the shadow agent keeps controlling the load balancing and the same remaining (Fig.1).

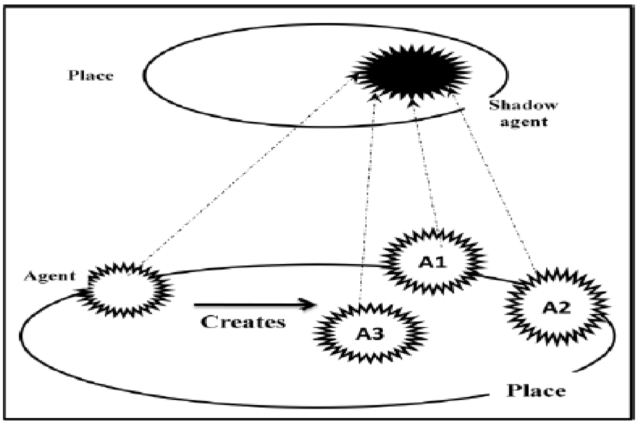

Fig.1 creating new agents

By removing a shadow agent all relied agents are affirmed orphans. It guarantees that orphan detection has removed all agents. Adding the path concept to this approach allows more control of the mobiles agents. Agent proxies maintain the relationship between the shadow and its dependent agents, (that fortifies the use of the shadow agent like a load balance controlling) thus create a path that leads to the agent. Storing the last place of the agent allows us to find the beginning of a path for each agent. Even though the path is lost, the agent will need to contact the shadow when the ttl becomes zero [5].

If an agent arrives at a place that has no proxy for the shadow (Fig. 2), one is created to retrieve the result of distributed computing.

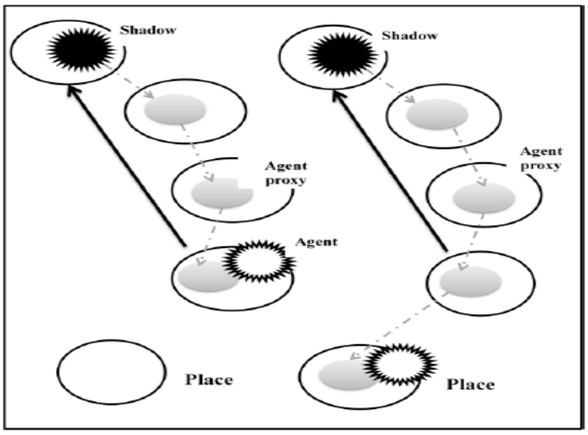

Fig. 2 the use of proxy for the recuperation of the agent path

The agent's destination node is stored in the proxy together with its ttl, when it migrates to another node. When the time to live becomes 0, the agent sends a request of the ttl to the agent's shadow to extend its life; therefore, the agent's new place is made known to the Shadow. The fact of knowing the location of agents allows knowing the amount of charge in each node.

The agent's place decrement in regular intervals it's time to live until it becomes 0. At this time, a message containing the ID of mobile agent and shadow is returned to the shadow agent. At the same time, a timer is triggered with a timeout and the agent enters the verification period. Each shadow has a particular timeout with the aim of more flexibility. However, this corrected by entering a per-place timeout. The chosen timeout is the minimum of agent and place timeout.

Once the home place of the shadow receives the check message, it stops the timer that has been started previously, and then the time to live is sent back to the concerned agent. This allows detecting terminated agents. The agent requests ttl from the responsible shadow. This latter verifies the ttl if it is superior to 0. if it is so the system sends it back to the requesting agent. When the agent receives the message, the timer for the timeout stops, and the agent's ttl is set. This ends the agent's check phase and allows it to continue its work again. When an agent arrives at a place, the agent proxies are searched. If none exists, a new one is created, and the agent gets a reference on it. As soon as an agent wants to depart, its ttl is checked. This is done to prevent an agent who is in the check phase to move around. If it is not in the check phase, the information in the agent proxy is updated to point to the target place. At the same time a timer is started, that removes the path after the addition of remaining ttl and timeout [5].

The shadow can decide whether an agent's lifetime is to be extended or not and by which interval. This decision will be clearly pored over when we discuss it in the load balancing by giving the shadow agent the possibility of giving the agent the order to migrate to the under loaded nodes.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

VI. OUR CONTRIBUTION:

A. How to estimate the nodes load:

To estimate workload of a particular node of the system, CPU utilization of the node is the measure used. Threshold policy is used to decide whether a node is lightly loaded or heavily loaded. It can be static where each node has a predefined entry value depending on its processing capability. Whereas in dynamic the threshold value of a node is calculated Location policies [1]

B. The agent's execution location:

At the start, the system stores in a database the information on the load on all nodes and the overall system load by creating a Receiver agent at every node of the network (fig. 3), The receiver agent is responsible for sending information about resources and the node load in which it was created (Fig. 4). The receiver agent like other mobile agents also asks to continue the ttl existed allowing the system to maintain its base of support provided on each machine update, which allows us to have a very flexible system change rapid weight, loads on the network.

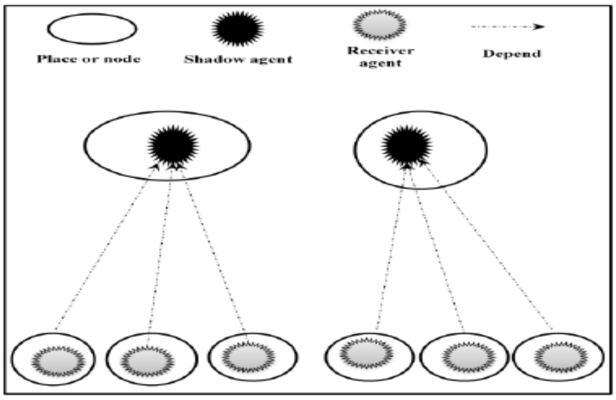

Fig.3 Receiver agent

```
public void sendLoadCPU(Place place){
        OperatingSystemMXBean bean =
ManagementFactory.getOperatingSystemMXBean();
        if (bean == null)
            throw new
NullPointerException("Unable to collect
operating system metrics, jmx bean is null");

        ACLMessage msg = newMsg(
ACLMessage.QUERY_REF );
        String receiverLocation =
this.currentPlace.getAddress();
        String content =
String.valueOf(bean.getAvailableProcessors())+""
+String.valueOf(bean.getSystemLoadAverage())+""+
receiverLocation;
        msg.setContent( content );
              msg.addReceiver( place.getAID());
              send(msg);
}

ACLMessage newMsg( int perf)
       {
         ACLMessage msg = new ACLMessage(perf);
         return msg;
       }
```

Fig.4 function of the Receiver agent used in the application that allows the recuperation of the CPU load

Then a shadow agent will be created in the same place of the node that contains the application source. It will also use the information stored in the database, in which the networks nodes load information have been together since the receptor agents managed the agent's migration to the least loaded neighbor nodes.

Each time an agent wants to redistribute a new load; it sends a message to the shadow agent stating its intention to create new agents (Fig. 5). Then, the shadow agent sends a response containing the ttl assigned to the new agent and creates nodes that contain best offers in terms of availability of resources it needs to send an established agent to be executed.

```
public void receiveCheckCreatAgents(Location
from,AID shadowId,AID agentId,Location target )
{
       if (timeToLive > 0)
       {
shadow = shadowList.find(shadowId);
timeToLive = shadow.timeToLive(from, agentId);
Agent createdAgent = new Agent();
int newTimeToLive = shadow.timeToLive(from,
createdAgent.getAID());

startTimer(timeToLive +
shadow.getTimeOut(agentId), shadow, agentId);

if(newTimeToLive>0)
    {
       startTimer(newTimeToLive +
shadow.getTimeOut(createdAgent.getAID()), shadow,
createdAgent.getAID());
    createdAgent.doMove(target);
    }
       }
}
```

Fig.5 the function treating the ttl demand for created agents

The principle of our solution lies in the fact that the request message ttl is accompanied by information on the implementation status of the agent, which is the node load from which the request was sent. Whenever the shadow agent receives a ttl request, it verifies the execution state of the agent. If the agent has already finished its task the shadow renews the ttl. If not, if the agent is awaiting execution, the shadow seeks in the database all the neighbors of the node that contains the agent that sent the request and compares ttl node load with the load of each neighboring node. If it finds that the node that contains the agent is less loaded it renews the ttl. If not he accompanied the ttl by an order of migration to the neighbor node with the lightest load (Fig. 6).

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

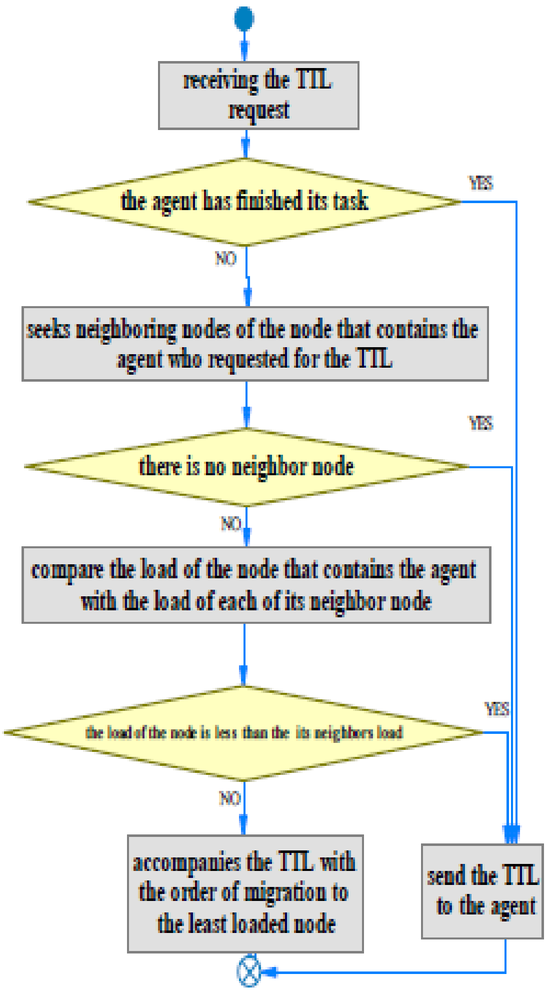

Fig.6 processing algorithm of ttl demand

C. When the agent will change the place:

The case of load balancing applications requires that the application should be restored exactly as it was before the movement of agents, for it must be transparent to the application itself. This requires a strong mobility mechanism [11], which in turns recommends the execution state to be transferred and resumed at the destination node. However, some recent works have shown that peculiar object-oriented support structures can allow load balancing by means of object migration mechanisms which do not require strong mobility [1]. The advantage of our approach is that it works with both mechanisms- since mobile agents can keep their states of execution.

To detect neighbor nodes of the node that contains the receiver agent diffuses into the network a CheckNeighbor message. Each time the CheckNeighbor message goes through a node the nodes number counter of traversed nodes will be incremented by 1. When the Receiver receives the agent CheckNeighbor message, it verifies the number of nodes that the message has traversed if it is equal to 1, it responds by ImYourNeighbor message containing the IP address of the node that contains it. Once the Receiver agent receives the response it transferred to shadow Home Place after adding the IP address of the node that contains it. The Shadow Home Place handles up to insert received data in the given database. Like that, we have the list of all neighboring nodes of the network. This database will be used by the given shadow agent to determine the most suitable node to receive the newly created or migratory agents (*Fig. 7*).

D. *Tasks performed by different agents:*

Once the stain of an agent is accomplished, it must join the shadow agent to provide the result of the task accomplished. However, in case an agent takes a long time to terminate its task, the system will need to control its state. In doing so, it has to find its location; this can be done with the use of the information stored in the agent proxies. If the agent is in the local list of active agents, it is already found. If not, the related agent proxy is searched. If it is not found, an error is returned. If it is discovered, a find request is sent to the target found in the proxy. At the target place the list of active agents is again examined. If the agent is found, a message containing its location is sent back. If not, the related agent proxy is searched again. If no proxy exists, an error is sent back. Otherwise, the message is sent on. This is repeated until the agent is found or the path ends (see Fig. 8).

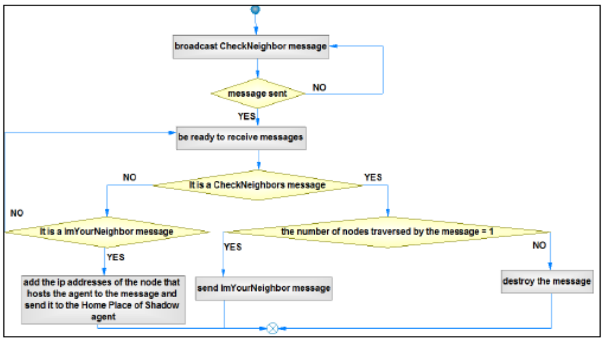

Fig.7 algorithm for the detection of neighboring node.

```
public String find(AID agentId)
        {
        if (agentList.find(agentId) != null)
        return(currentPlace.getID());
        else if(shadowList.find(agentId) != null)
        {
        sendFind(this, currentPlace, agentId);
        return(currentPlace.getID());
        }
        return null;
        }


public void sendFind(Location place, Location
currentPlace,AID agentID)
{
        ACLMessage msg = newMsg( ACLMessage.QUERY_REF
);
        String content = place.getName()+""+
currentPlace.getName() +""+ agentID.getName() ;
        msg.setContent( content );
        agentProxy.send(msg);
}
```

Fig.8 Finding Agents function used in the application

VII. APPLICATION

To illustrate our solution with an example, we have chosen JADE for its qualities as it is an open source; it is compliant with the FIPA (the Foundation for Intelligent Physical Agents) and can be compiled for devices with limited resources. JADE must be installed on all nodes. In our example, we have installed it on tree machines including a physical machine and the two others on VMWARE (Fig.9) and (Fig.10). Agents will be created on the node that is the least efficient and the system will take care of balancing the load on other nodes.

*Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication*

Node number	Number of processors/ processor speed	RAM	Hard disk
1	1/2GHz	256 MB	20 GB
2	2/2GHz	512 MB	20 GB
3	1/2GHz	1 GB	20 GB

Fig.9 used nodes configuration

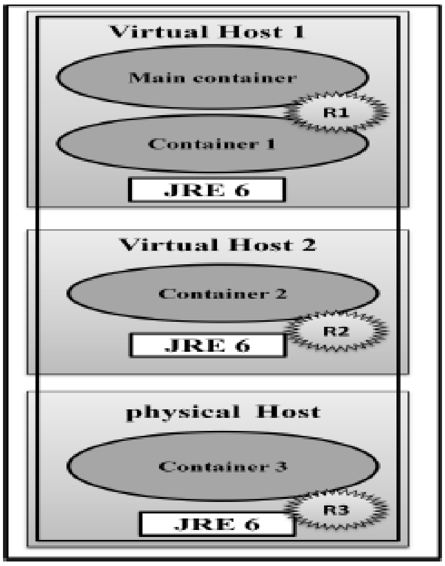

Fig.10 JADE Agent Platform distributed over several containers used in the application

We have chosen the example of buyers and Sellers, which is integrated with JADE. At first, all Sellers and Buyers agents are created in the less high-performance node. Sellers agents must communicate with each other, and they should have a high speed of execution to meet the demands of parallel Buyers agents and vice versa. Where does the need to use a load balancing approach to accelerate their response? In our application, we have used three Sellers and 60 buyers to implement the load balancing approach using shadow agent. While swinging the load on various nodes of the network, the system will execute the Sellers Agents in nodes providing the best deals in terms of resources.

Buyers Agents request quotes from Sellers Agents then proceed to buy at the cheapest price, if it is within the budget, otherwise they require a different set of quotes. Sellers respond to requests with a price that is valid for a limited time. The Sellers keep an eye on the price they quote to each customer and accept a purchase if the price offered is equal to the sum quoted and also if the request is received in time. Sellers must be able to manage several parallel inquiries (Fig.11).

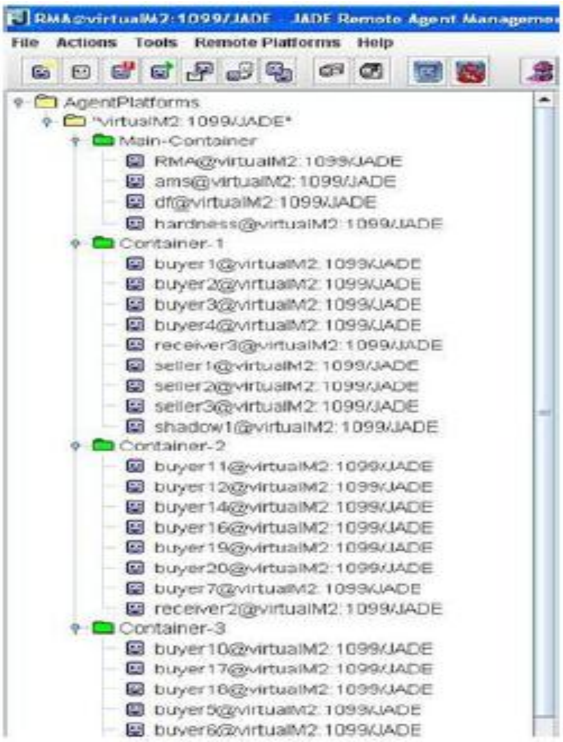

Fig.11 screenshot of the application executing in eclipse

We will create a special agent for the application execution time recovery, which is possible by the recovery system date of its creation and its destruction (Fig.12).

```java
public class Hardness extends Agent
{
long applicationStarTime =
System.currentTimeMillis() ;

public long getApplicationHardness (){
      long applicationEndTime =
System.currentTimeMillis() ;
      long hardness = applicationEndTime -
applicationStarTime;
      return hardness;
}
}
```

Fig. 12 recovery execution time part code

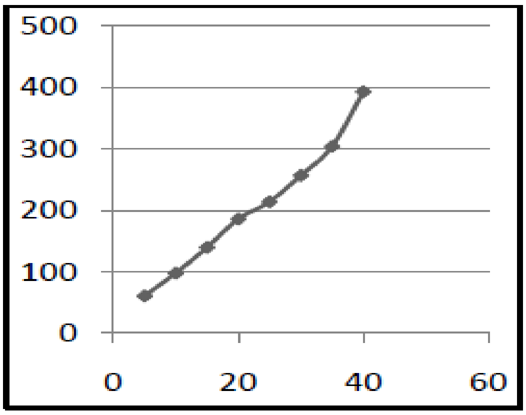

Fig.13 Application hardness according to the number of Buyers agents without Shadow approach

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

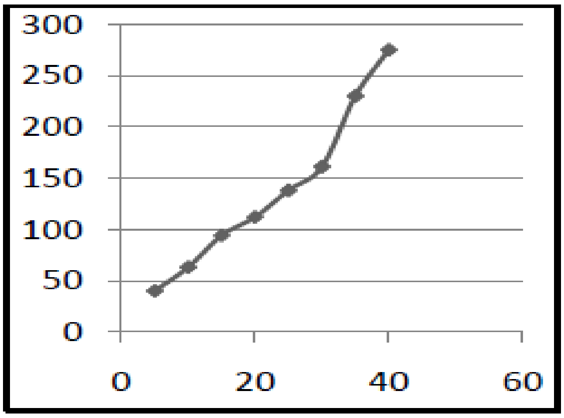

Fig.14 Application hardness according to the number of Buyers agents using the Shadow approach

We note that the execution time of the application has been reduced by integrating the results of the shadow agent load balancing approach in the application. The results of the shadow approach are more interesting, especially when the number of agents increases.

VIII. CONCLUSION

The more that load balancing by controlling mobile agent significantly reduces application's tasks execution time. It also allows them total control. The force of the load balancing by using mobiles agents is reflected in the made that agents may create other mobiles agents and assign to them a parts of tasks at any time of the application execution, which improves balancing the use of the network resource. The approach can be further expanded by improving the communication time between agents, which are still a subject of considerable research.

REFERENCES

1. R. Jadhav, S. Kamlapur I. Priyadarshini, "Performance evaluation in distributed systems, using dynamic load balancing" in International Journal of Applied Information Systems (IJAIS), Foundation of Computer Science FCS, pages 36-41 February 2012.
2. H.A. James, K.A. Hawick_and P.D. Coddington, "Scheduling Independent Tasks on Metacomputing Systems", Distributed and High Performance Computing Group, 9 March 1999.
3. N. Spanoudakis, P. Moraitis, "Modular JADE Agents Design and Implementation using ASEME" in IEEE International conference on web intelligence and intelligent agent technology, pages 221-228, 2010.
4. P. Moraitis and N. Spanoudakis, "The Gaia2JADE Process for Multi-Agent Systems Development", Applied Artificial Intelligence Journal 20(4-5), pages 251-273, 2006.
5. J. Baumann, K. Rothermel, "The Shadow Approach, an Orphan Detection Protocol for Mobile Agent", July 1998.
6. A. Singh, "An Efficient Load Balancing Algorithm for Grid Computing using Mobile Agent" in Anand Singh / International Journal of Engineering Science and Technology (IJEST), pages 4744-4747, 6 June 2011.
7. J. Stender, S. Kaiser, S. Albayrak, "Mobility-based Runtime Load Balancing in Multi-Agent Systems" 18th International Conference on Software Engineering and Knowledge Engineering, Reedwood City, CA, USA
8. J. Chakravarti, G. Baumgartner, M. Lauria, "The Organic Grid, Self-Organizing Computation on a Peer-to-Peer Network" IEEE TRANSACTIONS ON SYSTEMS MAN AND CYBERNETICS, 2005.
9. K. Rothermel, M. Schwehm, "MOBILE AGENTS", in Encyclopedia for Computer Science and Technology, 1998.
10. P. Sinha, "Distributed Operating Systems Concepts and Design", IEEE Computer Society Press.
11. G. Cabri, L. Leonardi, F. Zambonelli, "Weak and Strong Mobility in Mobile Agent Applications".
12. S. Fricke, K. Bsufka, J.Keiser, T. Schmidt, R. Sesseler, and S. Albayrak. "Agent based telematic services and telecom applications". In communications of the ACM, 2001.
13. N. Suri, J. Bradshaw, T. Groth, R. Breedy, A. Hill and R. Jeffers. "Strong mobility and fine-grained resource control in NOMADS" in Fourth international symposium on Mobile Agents, 2000.
14. J. Baumann, F. Hohl, N. Radouniklis, K. Rothermel. "Communication Concepts for Mobile Agent Systems" in Mobile Agents springer-Verlag, pp. 123 - 135, 1997.
15. J. Baumann, N. Radouniklis. "Agent Groups for Mobile Agent Systems", in Distributed Applications and Interoperable Systems, 1997.
16. F. Bellifemine, G. Caire, T. Trucco, G. Rimassa. "JADE PROGRAMMER'S GUIDE", last update: 08-April-2010. JADE 4.0
17. J. Cao, Y. Sun, X. Wang, S. Das, "Scalable Load Balancing on Distributed Web Servers Using Mobile Agents".
18. F. BOUZERAA, "Agents Mobiles et Systèmes Distribués" 14 December 2009.
19. A. Outtagarts, "Mobile Agent-based Applications: a Survey", in International Journal of Computer Science and Network Security, VOL.9 No.11, November 2009.
20. K. Chow, Y. Kwok, H. Jin, Kai Hwang "Comet: A Communication-Efficient Load Balancing Strategy for Multi-Agent Cluster Computing"

AUTHORS PROFILE

Bahaj Mohamed, was born in 1964, in Ouezzane, Morocco. He got his PhD in Applied Mathematics, from University of Pau, France, in 1993. He is now working as a Professor at the Department of Mathematics & Computer Sciences, University of Hassan 1er, Faculty of Sciences & Technology of Settat, Morocco. His research interests include pattern recognition, Load Balancing & Controls of mobiles agents, Semantic web & Ontology in MAS.

Soklabi Abdellatif, was born in 1985, in El JADIDA, Morocco. He had a license degree in computer engineering in 2009 and a master's degree in computer systems and networks in 2011. Now he is a PhD researcher in mobiles agents and web services in Department of Mathematics & Computer Sciences, University of Hassan 1er, Faculty of Sciences & Technology of Settat, Morocco. His research interests include, Load Balancing & Controls of mobiles agents, Interoperability between different MAS.

Cherti Ilias, was born in 1963, in Oujda, Morocco. He got his PhD in Applied Mathematics, from University of Rabat, Morocco, in 2007. He is now working as a Professor at the Department of Mathematics & Computer Sciences, University of Hassan 1er, Faculty of Sciences & Technology of Settat, Morocco. His research interests include Semantic web & Ontology in MAS, Controls of mobiles agents.