A Multi-agent Based Load Balancing Architecture to Support INPE Satellite Control Software

Andreia C. de Aquino*, Adriana C. Biancho†, Mauricio G. V. Ferreira‡, José Demisio S. da Silva§
National Institute for Space Research, São José dos Campos, SP, 12227-010, Brazil

A very important issue in distributed systems is the management of load among system nodes. Load balancing enables a better use of the system resource capabilities and a better performance by allocating nodes that are more suitable for the execution of some tasks. This work proposes a multi-agent load balancing architecture for distributed object applications called MALBA which uses artificial neural networks and a set of policies to support the process of migrating and replicating the application objects. MALBA architecture outlines a decentralized balancing process that balances the load among nodes which execute a distributed object application.

I. Introduction

THE availability of low-price microprocessors and the progress of the communication technology have increased the interest in distributed systems. The main advantages of these systems are high performance, availability of resources, and extensibility at a low cost. Taking into account these advantages, many organizations have adopted distributed systems and the use of distributed object applications.

Distributed objects are independent programs that might be located at any node of a network and might be accessed by remote clients via method invocation. Clients do not need to know where objects are located, i.e., if objects are located at the client's node or not[1].

In a distributed system, service requests arrive randomly at the nodes. This might generate a non-balanced state of the system that may lead to the existence of overloaded nodes and idle ones. This situation may harm services response time by the application objects, as well as, the usage of system resources[2].

The existence of a load balancing service avoids some nodes to get overloaded while others become idle. In this paper, the main concern is how to distribute the application objects among nodes for decreasing the application execution time and for optimizing the use of system resources.

Thus we propose a load balancing architecture called MALBA – **M**ulti-**A**gent **L**oad **B**alancing **A**rchitecture for Distributed-Object Applications, which is formed by different agents working together to provide a balanced solution for the system. Agents feel the environment and act on it by migrating objects from overloaded nodes to idle nodes and also by replicating highly required objects at less busy nodes in order to get a more balanced distribution of objects and consequently balancing the system load.

The difference between replicating and migrating an object is that in replication the object is kept at its local node and a copy of it is instantiated at a remote node whereas in migration the object is removed from its local node and it is instantiated at a remote node.

We propose some policies for the load balancing process, which are performed by the agents of MALBA architecture. An artificial neural network based agent classifies the system nodes in load levels. An agent makes decision as to balance the system by statistically analyzing the load levels of nodes. The selection of objects to migrate and the nodes to receive these objects is supported by migration policies which are performed by an agent. For the replication of objects some policies are used to guide the selection of objects to be replicated and the nodes

* Doctorate student, Applied Computing Postgraduate Program (CAP), Av. dos Astronautas 1758 Jd. Granja, ancarnie@lac.inpe.br.
† Doctorate student, Applied Computing Postgraduate Program (CAP), Av. dos Astronautas 1758 Jd. Granja, adcarnie@lac.inpe.br.
‡ Doctor Researcher, Satellite Control and Tracking Center (CRC), Av. dos Astronautas 1758 Jd. Granja, mauricio@ccs.inpe.br.
§ Doctor Researcher, Applied Mathematics and Computing Associated Laboratory (LAC), Av. dos Astronautas 1758 Jd. Granja, demisio@lac.inpe.br.

that will receive the replica. An agent performs the replication of the objects and an agent removes the replicas that are not in use anymore.

MALBA architecture follows a decentralized approach that means every node might take balancing decisions, thus implying in a non-centralized balancing control. As each node has autonomy to do the balancing, if a failure occurs at one node that does not put in risk the load balancing service.

This paper is organized as following: Section II introduces general features of MALBA architecture. Subsection A describes the load checking service, and in Subsection B and C we present the object migration and replication policies, respectively. In Subsection D and in Section III we describe the load balancing execution service and some results, respectively. In Section IV we present some related works and we draw some conclusions in Section V.

II. MALBA Architecture

Figure 1 illustrates MALBA architecture which comprises four services: the system load checking service, the load balancing execution service and the services related to the policies for migrating and replicating the application distributed objects.

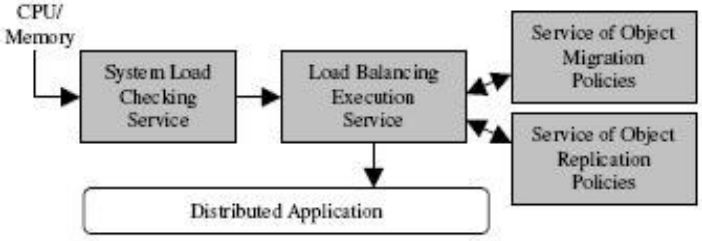

Figure 1. MALBA load balancing architecture.

In the following subsections we detail MALBA architecture services.

A. Load Checking Service

The load balancing is activated at random and every system node has autonomy to do so. The first step of the balancing process is to perform the Load Checking Service which is responsible for classifying the nodes in load levels and also for checking if there is a need to do the balancing. Two agents are in charge of performing this service: the Load Classifier Neural Agent and the System Load Checker Agent.

Load Classifier Neural Agent – the Neural Agent is responsible for stating the load level of the system nodes and it is present in all nodes. This agent collects two load indexes from a node: CPU usage and memory usage, and classifies this node in one of the following levels – (level 1): strong idle; (level 2): idle; (level 3): normal load; (level 4): overload; and (level 5): strong overload. For instance, the set {3, 1, 5, 4, 2} represents the possible load levels of a system with five nodes.

The Load Classifier Neural Agent uses an MLP neural network (Multiple Layer Perceptron) for classifying the load level of the system nodes. Our neural network was trained in a supervisioned way by the backpropagation algorithm[3]. Further details on our implementation is described in Section 3.

System Load Checker Agent – it has as its input the load level information generated by the Neural Agent. The Load Checker Agent checks whether the system is load balanced or not. Just in case the system is not balanced, it activates the balancing process.

First, the Load Checker Agent calculates the standard deviation (σ) considering the load level of all the system nodes, as follows:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (nc_i - \overline{nc})^2}{n-1}} \tag{1}$$

where:
- nc_i represents the load level of node i (provided by the Load Classifier Neural Agent);
- $\overline{nc}$ represents the load level average of the n nodes of the system;
- n represents the amount of nodes in the system.

Once the Load Checker Agent has calculated the standard deviation (σ), it applies the function $bal(\sigma)$ which defines whether the balancing process will be activated or not according to an activation threshold. Function $bal(\sigma)$ is as follows:

$$bal(\sigma) = \begin{cases} 1 & if \quad \sigma > \frac{\sigma_{\max}}{4} \\ 0 & otherwise \end{cases} \tag{2}$$

where:
- n represents the amount of nodes in the system;
- $\sigma_{\max}$ represents the maximum value of the standard deviation for n nodes.

If the result of the standard deviation (σ) is greater than ¼ of the standard deviation maximum value $\sigma_{\max}$, then the load balancing process will be activated, otherwise it will not.

The standard deviation maximum value $\sigma_{\max}$ is calculated considering all the nodes of the system (n nodes) and the threshold value to activate the load balancing process represents ¼ of this value, i.e., $\sigma_{\max}/4$.

The activation threshold was defined empirically taking into account the fact that as closer to zero the result of the standard deviation is, more balanced is the load of the overall system, and as the standard deviation gets far from zero, more non-balanced the overall system is and, consequently, there might be a need to activate the load balancing process.

B. Service of Object Migration Policies

Once the Load Checking Service has identified the need to activate the load balancing process, the distributed application, running at that moment, will have its object reallocated to better balance the overall system load.

Objects located at overloaded nodes should be migrated to idle nodes. The Service of Object Migration Policies is responsible for performing the choice of objects to migrate and the choice of nodes to receive these objects. These two selection processes are performed by the Migration Object Selector Agent and by the Migration Node Selector Agent, respectively.

Migration Object Selector Agent – The first action of the Object Selector Agent is to select the load dispatcher node ($N_{dispatcher}$), i.e., the most overloaded node in the system:

$$N_{dispatcher} = \arg\max_i \{nc_i\} \qquad i = 1,2,3,...,n \tag{3}$$

where:
- nc_i represents the load level of node i (provided by the Load Classifier Neural Agent);
- n represents the amount of nodes in the system.

Once a load dispatcher node ($N_{dispatcher}$) has been selected, the Object Selector Agent selects among its objects the ones which are not in use by the running application, i.e., the objects with no active connection, that is, objects with null connection, which are candidate objects to migrate because an object can only be migrated if it is not executing any of its services.

In the present discussion, C will denote the set of candidate objects to migrate and C_i will denote an object of this set.The set of candidate objects to migrate (C) is determined by the Object Selector Agent as follows:

$$C = \arg\underset{i}{Con}(\{ON_i = 0\}) \qquad i = 1,2,...,n \tag{4}$$

where:
- n represents the amount of objects in the $N_{dispatcher}$ node;
- ON_i represents the object i of $N_{dispatcher}$ node;
- $Con(\{ON_i\})$ represents the connection of object ON_i.

Once the Object Selector Agent has identified the candidate migrating objects (C), the next step is to quantify, for each candidate object, the following features:
- the size of the candidate object (in bytes);
- the amount of relations between the candidate object and the $N_{dispatcher}$ node objects;
- the amount of relations between the candidate object and the objects remote to $N_{dispatcher}$ node.

A relation between two objects occurs when one of the objects calls for a service provided by the other. If an object (*obj1*) relates with n other objects, the amount of *obj1* relations is n.

Among the candidate objects, the Object Selector Agent identifies an object that satisfies the criteria listed below.

(i) the object which is the biggest in size (number of bytes);
(ii) the object with the least amount of relations with the $N_{dispatcher}$ node objects;
(iii) the object which has the greatest amount of relations with the objects located remotely from the $N_{dispatcher}$ node.

The Object Selector Agent selects the migrating object as the one that satisfies the greater amount of the criteria listed before.

(i)
The Object Selector Agent identifies, from candidate objects of set C, the object (denominated i^{ob}) which has the biggest size, as follows:

$$i^{ob} = \arg\max_i (Tam\{C_i\}) \qquad i = 1,2,...,n \tag{5}$$

where:
n represents the amount of objects of set C;
$Tam\{C_i\}$ represents the size of object C_i.

(ii)
Among the candidate objects in set C, the Selector Agent identifies the object (denominated j^{ob}) which has the least amount of relations with the $N_{dispatcher}$ node objects, as follows:

$$j^{ob} = \arg\min_i (RI\{C_i\}) \qquad i = 1,2,...,n \tag{6}$$

where:
- n represents the amount of objects of set C;
- $RI\{C_i\}$ represents the amount of object C_i relations with objects of set ON.

(iii)
The Object Selector Agent identifies, from candidate objects of set C, the object (denominated k^{ob}) which has the greatest amount of relations with the objects located remotely from $N_{dispatcher}$ node, as follows:

$$k^{ob} = \arg\max_i (RE\{C_i\}) \qquad i = 1,2,...,n \tag{7}$$

where:
- n represents the amount of objects of set C;
- $RE\{C_i\}$ represents the amount of object C_i relations with objects remote to $N_{dispatcher}$ node.

Once i^{ob}, j^{ob}, and k^{ob} have been defined, the Object Selector Agent creates a histogram for each object in the $N_{dispatcher}$ node representing the object frequency in set $\vartheta = \{i^{ob}, j^{ob}, k^{ob}\}$.

The set ON represents all the objects in $N_{dispatcher}$ node and considering $r \in ON$, the histogram H_r is defined as follows:

$$H_r = \sum_{q=1}^{3} 1_r(\vartheta(q)), \qquad 1_r = \begin{cases} 1 & if \quad \vartheta(q) = r \\ 0 & otherwise \end{cases} \tag{8}$$

The Object Selector Agent finally defines the object to migrate ($O_{migrate}$) as follows:

$$O_{migrate} = \arg\max_{t}(\{H_t\}) \qquad t = 1,2,...,n \tag{9}$$

where:

- n represents the amount of objects in $N_{dispatcher}$ node.

Migration Node Selector Agent – Once the object to migrate ($O_{migrate}$) has been selected, the Node Selector Agent chooses the most suitable node to receive this object.

The first step of the Node Selector Agent is to select the node which has the least load in the system, based on the load level of nodes. In case there are nodes with the same least load level then a decision criterion is used to select the node to receive the object $O_{migrate}$ as the one with the greastest amount of relations with the object $O_{migrate}$.

C. Service of Object Replication Policies

According to Ferreira[4], object replication benefits the load balancing of the overall system because it can relieve the load of a node by replicating its highly requested object at another node, implying that new service requests will be answered by the replicated object.

The Service of Object Replication Policies is responsible for leading the selection of the object to replicate and the selection of the replica receiver node. These selections are performed by the Replication Object Selector Agent and by the Replication Node Selector Agent, respectively. The former selects from the candidate objects (which are the objects in the most overloaded node that have active connections, i.e., the objects in use at the moment) the one which has the greatest amount of active connections to be the object to replicate. The node to receive this replica is selected by the latter agent as the one with the least load level in the system or in case there are nodes with the same least load level the one which has the greatest amount of active connections with the object to replicate.

D. Load Balancing Execution Service

The Load Balancing Execution Service is responsible for reconfiguring the distribution of objects of the application in order to get a more load-balanced system. This service is formed by the Object Migrator Agent, the Object Replicator Agent, and the Replica Remover Agent.

Object Migrator Agent – This agent is responsible for migrating objects from overloaded nodes to idle nodes.

This agent communicates with the agents of the Migration Policies Service in order to be aware of the object to migrate and the load receiver node. Based on these perceptions, the Object Migrator Agent performs the object migration.

Object Replicator Agent – This agent is responsible for replicating objects from overloaded nodes to idle nodes.

This agent communicates with the agents of the Replication Policies Service in order to be aware of the object to replicate and the replica receiver node. Based on these perceptions, the Object Replicator Agent performs the object replication. Once replication occurs, all the future requests to the replicated object will be answered by its replica, because the replica has higher priority than the original object.

Replica Remover Agent – This agent is responsible for removing object replicas, in overloaded nodes, which are no longer in use (connection = 0).

III. Tool and Results

A prototype tool is under development to realize the MALBA architecture project. We have currently implemented MALBA Load Checking Service which is executed by the Load Classifier Neural Agent and the System Load Checker Agent.

In order to implement the Neural Agent, we firstly practiced some knowledge engineering to associate CPU and memory usages to the five load levels defined in Subsection A. Table 1 presents these associations.

Table 1. CPU and memory usages associated to our five load levels.

	MEMORY				
CPU	0 – 0.19	0.2 – 0.39	0.4 – 0.59	0.6 – 0.79	0.8 – 1.0
0 – 0.19	Level 1	Level 2	Level 3	Level 4	Level 5
0.2 – 0.39	Level 2	Level 2	Level 3	Level 4	Level 5
0.4 – 0.59	Level 3	Level 3	Level 3	Level 4	Level 5
0.6 – 0.79	Level 4	Level 4	Level 4	Level 4	Level 5
0.8 – 1.0	Level 5	Level 5	Level 5	Level 5	Level 5

The Load Classifier Neural Agent uses an MLP neural network (Multiple Layer Perceptron) which has been trained in a supervised way by the error backpropagation algorithm[3]. The network architecture is formed of 02 entries, 37 neurons in the hidden layer and 03 neurons in the output layer, having as their activation function the hyperbolic tangent and the sigmoid logistic, respectively.

The training set had 10201 inputs and in 30000 epochs the network achieved an error of 0.000008.

The Neural Agent outputs, i.e., the load level of system nodes, are the entries to the Load Checker Agent which uses these values to statistically verify whether it is necessary to balance the system load or not. For further details on the Load Checker Agent behaviour see Subsection A.

After training the Neural Agent and coding the Load Checker Agent behavior, tests were conducted to validate MALBA architecture tool. Table 2 shows the specification of two test cases. The execution of these test cases produced the expected results presented in Table 2.

Table 2. Test cases executed for testing MALBA architecture Load Checking Service.

Test Case	Test Data			Expected Results	
	Node	CPU Usage	Memory Usage	Node Load Level	Balancing Decision
01	1	0.92	0.63	5	Activate load balancing
	2	0.31	0.27	2	
	3	0.78	0.64	4	
02	1	0.57	0.46	3	Do not activate load balancing
	2	0.71	0.58	4	
	3	0.53	0.37	3	

In Test Case 01 execution, the Neural Agent output was the following load levels: 5, 2, 4. The standard deviation of the system node load levels was `1.247219128924647` and the maximum standard deviation for this three-node system was `1.8856180831641267`. Thus, the balancing decision was to activate load balancing. In Test Case 02, the Neural Agent output was 3, 4, 3, the standard deviation was `0.4714045207910317` and the maximum standard deviation was `1.8856180831641267`. Thus, the decision was not to activate load balancing.

A relevant issue is that MALBA architecture considers that the system nodes are homogeneous, i.e. all the nodes are equivalent in their hardware and software architectures. This allows the application objects to be located at any system node. Besides that, nodes are considered to be equivalent in their processing and storage capabilities.

IV. Related Works

MALBA migration and replication policies consider communication costs related to local objects and also to remote objects. The idea is to locate objects that communicate to each other, i.e., objects that call other objects services, together in the same node in the system to reduce the remote communication costs. In Ref. 5 these policies are not defined and in Ref. 2 they are defined, however the authors consider just the communication costs among local tasks.

The load balancing service management is approached in a centralized way by some authors, as in Ref. 5 and Ref. 6. However, MALBA architecture approaches the problem in a decentralized way. A centralized management might put the load balancing service in risk because of the existence of a unique failure point in the system.

V. Conclusions

MALBA architecture has been designed to provide system load equilibrium by a better distribution of the objects in a distributed application. MALBA load balancing service is performed in a decentralized and dynamical way, redistributing the load among system nodes at execution time. This redistribution is based on a set of policies which leads the object migration/replication process. These policies are guidelines to select objects to migrate/replicate and also to select the recipient nodes of these objects.

In MALBA architecture, the load balancing is performed by means of agents that communicate to each other to take balancing decisions. This multi-agent architecture uses artificial neural networks, by means of a neural agent, to classify the load level of system nodes.

In order to validate MALBA architecture, we intend to apply the proposed ideas to a distributed object application such as the Prototype of INPE (National Institute for Space Research) Satellite Control System. Optimizing the use of INPE computational resources is a mean for the Control Satellite System to be improved to support additional new satellites, operating at computational conditions that avoid overloads.

Acknowledgments

This work was financially supported by Coordination of Postgraduate Personnel Improvement (CAPES).

References

[1]Purao, S., Jain, H. K., and Nazareth, D. L., "Effective Distribution of Object-Oriented Applications," *Communications of the ACM*, 41(8), 1998, pp. 100-108.

[2]El-Abd, A., and El-Bendary, M., "Neural-Based Selection and Location Policies for Dynamic Load Balancing in Distributed Computing Systems," *Proceedings of the IASTED International Conference on Modeling and Simulation*, 1998.

[3]Haykin, S., *Neural Networks – A Comprehensive Foundation*, Macmillan, New York, 1994.

[4]Ferreira, M. G. V., "A Dynamic and Flexible Architecture for Distributed Objects applied to Satellite Control Software," Doctorate Thesis in Applied Computing Postgraduate Program, INPE, São José dos Campos, Brazil, 2001.

[5]Yang, J., Jizhou, S., and Zunce, W., "Load Balance in a New Group Communication System for the WAN," *CCECE – CCGEI*, IEEE, Montreal, 2003.

[6]Alvim, R., Grossmann, F., and Dantas, M., "Implementação de uma Arquitetura para Serviço Distribuído com Grande Disponibilidade em Ambiente Linux," Revista Eletrônica de Iniciação Científica, Sociedade Brasileira de Computação, Vol. 2, No. 3, 2002.