

A Multi-agent Load Balancing Architecture for Distributed Object Applications

Andreia C.
de Aquino

Adriana C.
Biancho

Maurício G.V.
Ferreira

José Demisio S.
da Silva

National Institute for Space Research (INPE)
Avenida dos Astronautas, 1758 – São José dos Campos – SP – Brazil

Abstract – *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 denominated 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 which balances the load among nodes which execute a distributed object application.*

Keywords: load balancing, distributed object application, multi-agent systems, artificial neural networks.

1 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 same node of the client or not [1].

In a distributed system, service requests arrive randomly at the nodes. This might generate a non-balanced state of 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 denominated MALBA – *Multi-Agent Load Balancing Architecture 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 that perform the following tasks: (i) the classification of the system nodes in load levels; (ii) the checking of the need for balancing the system; (iii) the migration of objects; (iv) the selection of objects to migrate; (v) the selection of the load receiver node; (vi) the replication of objects; (vii) the selection of objects to replicate; (viii) the selection of the replica receiver node; and (ix) the removal of inactive replicas.

An artificial neural network based agent performs the task (i) classifying the system nodes in load levels. In task (ii), an agent makes decision as to balance the system by statistically analyzing the load levels of nodes. In task (iii), an agent performs the migration of objects. Tasks (iv), (v), (vii), and (viii) make use of defined criteria, denominated migration and replication policies, to select objects to migrate and to be replicated, respectively, and to select the nodes that will receive the migrated object and the replica. In task (vi), an agent performs the replication of the objects, and in task (ix) an agent removes the replicas, which are not in use anymore.

MALBA architecture follows a decentralized approach that means every node might take balancing decisions, having no centralized node controlling the balancing. As each node has autonomy to do the balancing, if a failure

occurs at one node this does not put in risk the load balancing service.

This paper addresses the following: Section 2 introduces general features of MALBA architecture. Subsection 2.1 describes the load checking service, and in Subsections 2.2 and 2.3 we present the object migration and replication policies, respectively. In the two remaining sections we describe the load balancing execution service and some related works. We draw some conclusions in Section 4.

2 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. Next we detail these services.

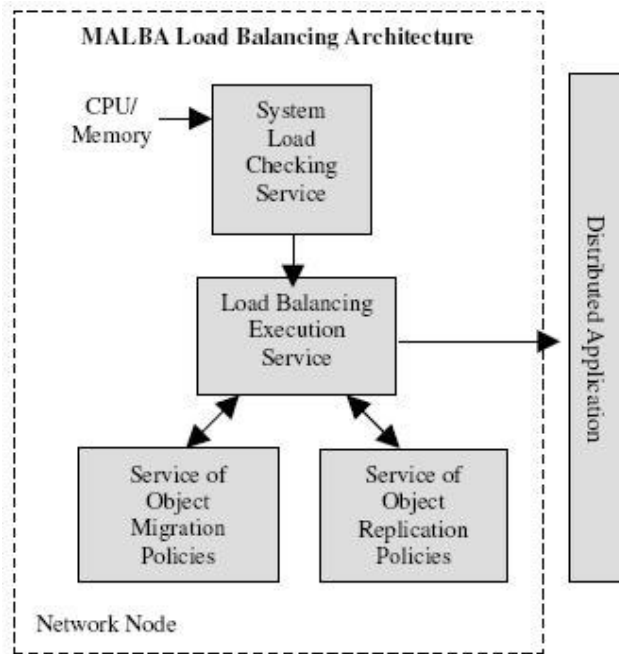


Figure 1. MALBA Load Balancing Architecture.

2.1 Load Checking Service

As every system node has autonomy to do the balancing, they do it at random. The first step to balance the system load is performing the load checking service which is responsible for classifying the load level of nodes 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, both presented in Figure 2.

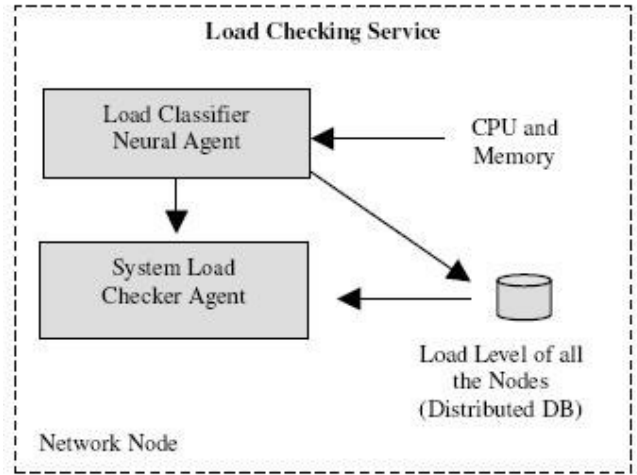


Figure 2. Agents of MALBA Load Checking Service.

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: UCP 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. Therefore, the set {1, 2, 3, 4, 5} represents the possible load levels of a system with five nodes.

The Load Classifier Neural Agent uses a MLP neural network (Multiple Layer Perceptron) for classifying the load level of the system nodes. Our neural network was trained in a supervised way by the backpropagation algorithm [3]. The neural agent output is written in a distributed database, from which MALBA agents share information.

The System Load Checker Agent has as its input the load level information generated by the neural agent. This 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:

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

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 a function denominated $bal(\sigma)$ which defines whether the balancing process will be

activated or not according to an activation threshold. Function $bal(\sigma)$ is as follows.

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

where:

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

If the result of the standard deviation (σ) is higher than $\frac{1}{4}$ of the standard deviation maximum value σ_{\max} , then the load balancing process will be activated, otherwise it will not.

The standard deviation maximum value σ_{\max} is calculated considering all the nodes of the system (n nodes) and the threshold value to activate the load balancing process represents $\frac{1}{4}$ 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 further from zero, more non-balanced is the overall system and, consequently, there might be a need to activate the load balancing process.

Figure 3 illustrates the activation of the load balancing process. At a random time, the Load Checker Agent calculates the standard deviation of the overall system and compares it with the activation threshold to take the decision whether to do the balancing or not.

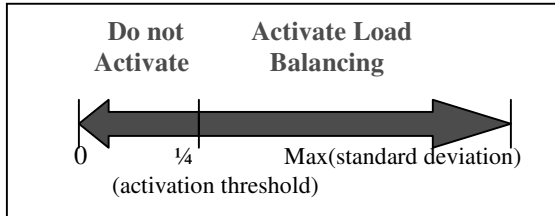


Figure 3. Load balancing activation in MALBA architecture.

2.2 Service of Object Migration Policies

Once the Load Checking Service identified the need to activate the load balancing process, the distributed application, running at that moment, will have its objects replaced 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 leading the choice of objects to migrate and the choice of nodes to receive these objects. These two selection processes are performed by the Object Selector

Agent and by the Node Selector Agent, respectively (Figure 4).

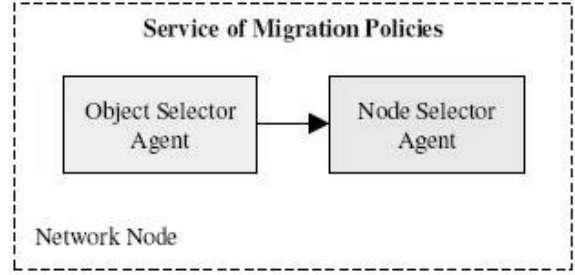


Figure 4. Agents of the Migration Policies Service in MALBA architecture.

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

$$N_{\text{dispatcher}} = \arg \max_i \{nc_i\} \quad i = 1, 2, 3, \dots, n \quad (3)$$

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

This set of candidate objects to migrate (C) is determined by the Object Selector Agent as follows:

$$C = \arg \text{Con}(\{ON_i = 0\}) \quad i = 1, 2, \dots, n \quad (4)$$

where:

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

Once the Object Selector Agent identified the candidate objects to migrate (C), 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 $N_{\text{dispatcher}}$ node objects;
- the amount of relations between the candidate object and the objects remote to $N_{\text{dispatcher}}$ node.

The meaning of a relation between two objects is that it occurs when one of the objects calls a service provided by the other object. Considering an object (*objl*) relates with *n* other objects, the amount of *objl* relations is *n*.

Among the candidate objects, the Object Selector Agent identifies, for each criterion listed below, an object that satisfies the criterion. This criteria set is:

- (i) the object which has the biggest size (in bytes);
- (ii) the object which has 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 set listed above.

(i)

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

$$i^{ob} = \arg \max_i (Tam\{C_i\}) \quad i = 1, 2, \dots, n \quad (5)$$

where:

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

(ii)

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

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

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*) which has the greatest amount of relations with the objects located remotely from $N_{dispatcher}$ node, as follows:

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

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:

$$H_r = \sum_{q=1}^3 1_r(\vartheta(q)), \quad (8)$$

$$1_r = \begin{cases} 1 & \text{if } \vartheta(q) = r \\ 0 & \text{otherwise} \end{cases}$$

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

$$O_{migrate} = \arg \max_i (H_i) \quad i = 1, 2, \dots, n \quad (9)$$

where:

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

Migration Node Selector Agent – Once the object to migrate ($O_{migrate}$) was 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 which have the same load level, and this level is the least, then a criterion of decision is used to select which node will receive the object $O_{migrate}$.

This criterion of decision consists of selecting the node which has the greatest amount of relations with the object $O_{migrate}$. The pseudo-code of the Node Selector Agent is as follows:

Function MIGRATION NODE SELECTOR AGENT
(perception: object to migrate)
returns receiver node

least-load-node \leftarrow select node(s) which has/have the least load

if least-load-node > 1
then receiver node \leftarrow select the node which has the greatest amount of relations with the object to migrate

2.3 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 called object to another node, what implies that new service calls will be answered by the replica and not by the original 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 Object Selector Agent and by the Node Selector Agent, respectively (Figure 5).

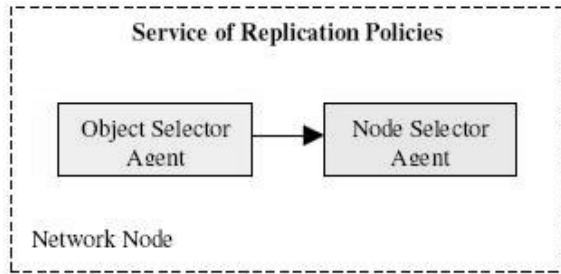


Figure 5. Agents of the Replication Policies Service in MALBA architecture.

Replication Object Selector Agent – This agent is responsible for selecting, in an overloaded node, an object to replicate to an idle node.

First, the Object Selector Agent identifies the most overloaded node in the system and then selects from its objects the ones which have an active connection, i.e., the objects in use at the moment. These objects are candidates for replication, reminding that objects with active connections cannot be migrated, just replicated.

Next, the Object Selector Agent selects from the candidate objects the one which has the greatest amount of active connections to be replicated, as shown in the pseudo-code that follows:

```

Function REPLICATION OBJ SELECTOR AGENT
  (perception: the most overloaded node)
  returns object to replicate

  candidate_objects ← select, from the most overloaded node,
                      objects which have active connections
  object to replicate ← candidate_object which has the greatest
                       amount of active connections
  
```

Replication Node Selector Agent – Once the object to replicate has been selected, the Node Selector Agent selects the most suitable node to receive the copy of this object.

The Node Selector Agent accomplishes this task by selecting the node which has the least load in the system, based on the nodes load levels. If more than one node has the same load level, and this level is the least in the system, then a decision criterion is applied to select which node is the replica receiver one.

This decision criterion consists of selecting the node which has the greatest amount of active connections with the object to replicate. The Node Selector Agent pseudo-code is as follows.

```

Function REPLICATION NODE SELECTOR AGENT
  (perception: object to replicate)
  returns replica receiver node

  least-load-node ← select node(s) with least load
  if least-load-node > 1
  then replica receiver node ← select the node which has the greatest
                               amount of active connections
                               with the object to be replicated
  
```

2.4 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 agents illustrated in Figure 6.

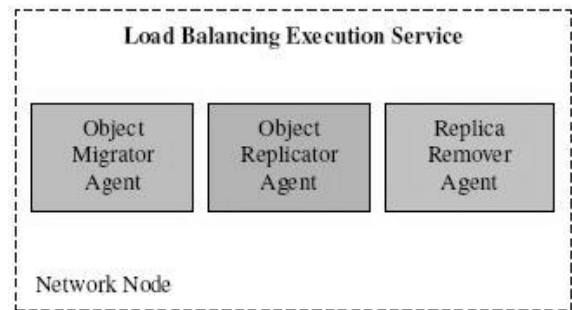


Figure 6. Agents of the Load Balancing Execution Service in MALBA architecture.

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 as shown in the following pseudo-code.

```

Function OBJECT MIGRATOR AGENT
  (perception: the most overloaded node,
   object to migrate {instance of classY},
   load receiver node)
  returns action of migrating object

  remove from the most overloaded node the object to migrate
  create in the load receiver node a new instance of
  classY: new-object ← new classY()
  
```

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 as shown in the pseudo-code below.

Once replication occurs, all the future calls to the replicated object will be answered by its replica, because it is given the replica a higher priority than the original object.

Function OBJECT REPLICATOR AGENT
(perception:
 object to replicate {instance of classZ},
 replica receiver node)
returns action of replicating object

create in the replica receiver node a new instance of
classZ: new-object ← new classZ()
assign to new-object a higher priority than the
original object of class Z

Replica Remover Agent – This agent is responsible for removing object replicas, in overloaded nodes, which are no longer in use (connection = 0). The Replica Remover Agent performs the replica removal as shown in the pseudo-code.

Function REPLICA REMOVER AGENT
(perception: the most overloaded node)
returns action of removing object

candidate_objects ← select objects with
connection = zero in the most overloaded
node

for each candidate_object **do**
 if amount of candidate_object in the system is
 greater than 1
 then remove candidate_object from the most
 overloaded node

3 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 at the same node in the system to reduce the remote communication costs.

In [1] these policies are not defined and in [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 [5] and [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.

4 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.

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 new space missions, operating at computational conditions that avoid overloads.

5 References

- [1] S. Puroo, H.K. Jain, and D.L. Nazareth, "Effective Distribution of Object-Oriented Applications", Commun. ACM 41(8), pp. 100-108, 1998.
- [2] A. El-Abd and M. El-Bendary, "Neural-Based Selection and Location Policies for Dynamic Load Balancing in Distributed Computing Systems", Proceedings of the IASTED International Conference on Modeling and Simulation, May 1998.
- [3] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, New York, 1994.
- [4] M.G.V. Ferreira, "A Dynamic and Flexible Architecture for Distributed Objects applied to Satellites Control Software", Doctorate Thesis in Applied Computing, INPE, São José dos Campos – SP, 2001.
- [5] J. Yang, S. Jizhou, and W. Zunce, "Load Balance in a New Group Communication System for the WAN", IEEE, CCECE – CCGEI 2003, Montreal, May 2003.
- [6] R. Alvim, F. Grossmann, and M. Dantas, "Implementação de uma Arquitetura para Serviço Distribuído com Grande Disponibilidade em Ambiente Linux", Revista Eletrônica de Iniciação Científica – Sociedade Brasileira de Computação, Vol. II, No. III, 2002.