A DYNAMIC ARCHITECTURE FOR RECONFIGURATION OF WEB SERVERS CLUSTERS

Carla Marques[1], Isabel Oliveira[2], Giovanni Barroso[2] and Antonio Serra[3]

[1]Universidade do Estado do Rio Grande do Norte (UERN)
BR 110 Km 46,59600-900,Mossoró-RN-Brasil
carla.katarina@gmail.com
[2]Universidade Federal do Ceará(UFC)
Campus do Pici, Fortaleza-CE, 60455 760 – Brasil
isabelregio@gmail.com, gcb@fisica.ufc.br
[3]Instituto Federal de Educ. Ciencia e Tecnologia
Av. 13 de Maio, 2081, Fatima, Fortaleza – CE, 60040-531 - Brasil
prof.serra@gmail.com

ABSTRACT

The allocation planning of resources in a web server cluster is accomplished nowadays by the administrator. Once the internet is quite dynamic, as far as the use of resources is concerned, such a task may be considered critical and inefficient if accomplished manually. Our approach benefits from the use of agents to learn from the environment and adjust automatically the behavior of the system to make a better use of the available resources. With this approach it is possible to help the administrator by minimizing your stress in moments of work overload. The conception, specification, adopted allocation planning strategy, modeling in Petri nets, implementation of this platform in the Java language are presented. Experimentations and simulations which prove the efficiency of the proposal are presented.

KEYWORDS

Cluster of Web server, Dynamic Reconfiguration, Multiagent system, Petri Nets

1. INTRODUCTION

In recent years, cluster computing technology has become a cost-effective computing infrastructure that aggregates effectively different types of resources (such as processing, storage, and communication resources). It is also considered to be a very attractive platform for low cost super-computing. Thus, a cluster of computers is easy to build and highly scalable. Basically, it consists of several workstations interconnected through a high-speed network for information exchange and coordination among them.

Even for experienced cluster administrators, the management of a cluster is an exhausting job as allocating the cluster's resources by hand can easily become unmanageable. This may occur because the needs of processing requirements can change very quickly in a dynamic environment such as the Internet. With this motivation, this paper presents a solution to increase the availability of services in clusters of web servers using a Multi-agent system. The objective of this architecture is to minimize the work of the cluster administrator and reduce the possibility of errors in periods of load peaks, because the load's Internet is unpredictable. The advantage of the agents' use is that this system [2] provides the dynamic management of resources because it is a solution inherently distributed.

Presented in this paper is a *Dynamic Architecture for Reconfiguration of Web servers Clusters* (DARC) that perform a self-reconfiguration of the resources in a web server to help the

DOI : 10.5121/ijcnc.2011.3308

administrator. We integrated this architecture to the platform of *Web Servers - Differentiated Services Admission Control* (WS-DSAC) [3].

The main contributions of this work are:

- Implementation of a self-reconfiguration of clusters, without the need of interference of the administrator;
- Dynamic reallocation of web servers executed by agents, based on information obtained through interaction with the WS-DSAC platform;
- Verification of the appropriate amount of web servers to be relocated between clusters, based on reconfiguration decisions of the agents.

The rest of this paper is organized as follows. In Section 2, we present some related works to put our work in context. In Section 3, we describe the WS-DSAC load balancing Platform, which the multi-agent system is integrated. In Section 4, we present our approach for dynamic reconfiguration. The architecture model in Colored Petri Net (CPN) and the results of the simulation model are presented in section 5. The implementation and analysis of the DARC architecture is presented in Section 6. Finally, we present the conclusions in Section 7.

2. RELATED WORK

In this section, we present some relevant works that have been developed in the area of load balancing.

The work presented in this paper builds upon [3], which presents the *WS-DSAC* platform, and so performs load balancing in a cluster of web servers. This platform relies on service differentiation to allocate available resources. Thus, the servers are grouped in different web clusters according to predefined service classes, and each cluster is responsible for processing requests from a specific service class with a certain *Quality of Service* (QoS). The QoS is measured through the concept of “reactivity coefficient", defined in [4] as a measure of the load on a server; more specifically, it is an estimate of the average waiting time of a task that must be executed (in our case, of a request that must be processed). Due to its importance for the work presented in this paper, we describe WS-DSAC in more detail in Section 3.

In [1], is presented the cluster based replication architecture for load-balancing in peer-to-peer content distribution systems using an intelligent replica placement technique.

Several works use agent technology to perform dynamic load balancing [5, 6, 7 & 8]. In these works there aren’t automatic reconfigurations of cluster. Thus, they require an administrator that manually inputs the configuration information. This manual work is annoying and error-prone, especially when the scale of clustering enlarges or the configuration changes dynamically.

Some works propose strategies for automatic dynamic reconfiguration. See [9], where a proprietary operating system called Fire Phoenix is defined; although it can be installed on top of another operating system, having two different kernels introduces an additional overhead. This work is closely related to ours as it also performs a reconfiguration of clusters using agents, aiming at providing a scalable and highly available distributed heterogeneous platform.

In the work presented in [10], an agent-based self-configuration mechanism is proposed, that allows for the automatic allocation of available resources to overloaded clusters without human intervention; however, this approach relies on a central server, which is a single point of failure. In another example [11], presents an approach to allocate a server in a cluster for the processing of a request and activate automatically standby servers when the cluster's load increases. Initially, a request is allocated to a server randomly. If this server cannot process the request, it is forwarded to another server, and so on, until one server is able to process it or the maximum

amount of time allocated for the request has been exceeded. This redirection-based approach can be inefficient. Moreover, no load-balancing is performed and a single cluster is considered.

The solution presented in this paper aims at a dynamic reconfiguration of clusters of web servers using multi-agent systems, minimizing the task of the Cluster Administrator. In this architecture, the agents learn from previous information to make the best reconfiguration in the future. This new architecture DARC is presented in section 4.

3. THE WS-DSAC PLATFORM: LOAD BALANCING

In this section the WS-DSAC Platform is presented, which the DARC Architecture is integrated. This platform has the main objectives: to balance the imposed load, to guarantee different QoS levels and to use available resources in an effective way.

The WS-DSAC platform (Figure 1) is composed by a set of elements: *Class Switch*, *Cluster Gateways* and *Web Server Nodes*. *Class Switch* is responsible of classification and admission control of client requests. It receives incoming HTTP requests, identifies the service class and sends each request to a specific *Cluster Gateway*. *Cluster Gateway* chooses a least loaded Web Server Node to process the requests sent by the *Class Switch*.

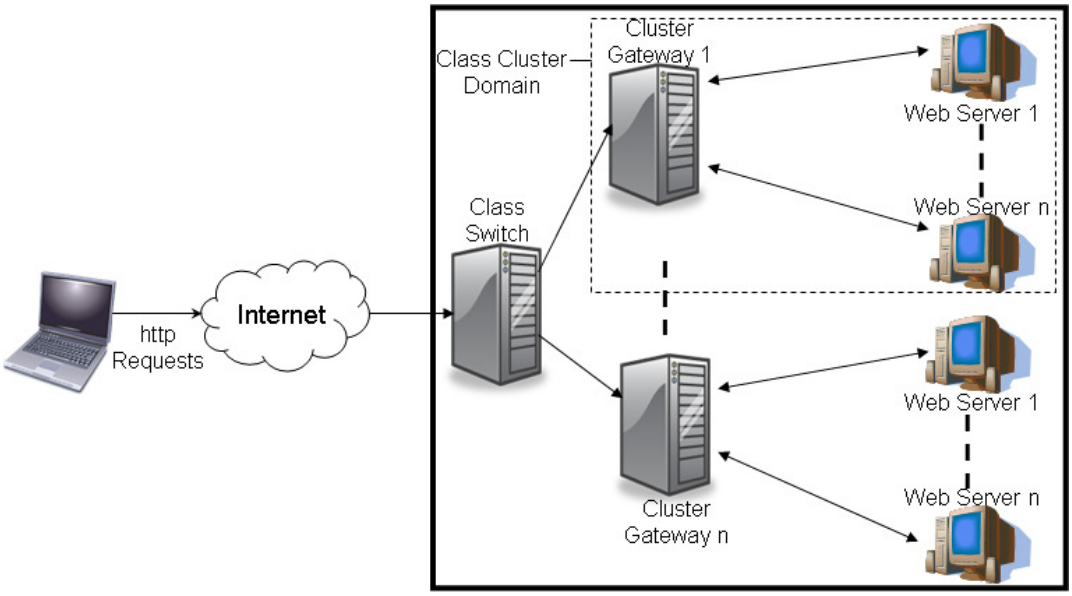

Figure 1. Overview WS-DSAC Platform

The platform offers different levels of QoS based on the differentiation of services. Services are deployed in a number of Web Server nodes and they are composed by Internet services and distributed objects. Incoming requests may belong to different classes of services. The platform administrator associates each class of service to a maximal load value that can be achieved by its "class cluster" domain. The RC parameter (reactivity coefficient) is associated with each pre-established class of services and this parameter is used in each Web server.

The strategy adopted in [3] for resources dynamic reallocation between service classes is based on "class clusters" work mode changes. During a time interval, cluster resources priority allocated for one specific service class can be in one of three possible states: *shared, exclusive* or *saturated*. When a specific "class cluster" works in the *shared mode*, it permits the respective class has available resources that can be used by other service classes. They can be used without compromise agreements established with the native class during a predefined time interval. In this mode the "class cluster" also receives incoming requests of other service classes. In the

exclusive mode, the "class cluster" only receives requests of its native class. This signifies that load levels have reached established thresholds and accepting new requests of other classes can cause rejection of native class requests. In the *saturated mode* the "class cluster" will accept no new requests because, if new requests are accepted, resources will not be sufficient to guarantee the QoS assured to requests that are been processed.

Work mode changes on the "class clusters" are based on two thresholds: a dynamic threshold recomputed at each time interval ρ_{ki} and R_{emk} and a threshold that limits the maximum RC of one specific class, R_{max} and $R_{ac,}$ where: ρ_{ki} estimates load average of servers registered on the class cluster "i"; R_{emk} establishes the RC value that can be reached by the class cluster; R_{max} is the maximum value allowed for R_{emk} where the cluster works in *shared mode* and R_{ac} is the limit value where the cluster works in *exclusive mode*.

When a request arrives on the platform the "class switch" identifies the request class. Given that a cluster is the least loaded, the "class switch" performs the hereafter algorithm: if $\rho_{ki} \leq R_{max}$ then the cluster works in shared mode; else if $\rho_{ki} > R_{max}$ and $\rho_{ki} \leq R_{ac}$ the cluster works in exclusive mode; else the cluster works in saturated mode.

The R_{max}, R_{ac} and ρ_{ki} variables, presented in this section, control the WS-DSAC platform. The two main limitations of the WS-DSAC platform presented are: 1) an administrator is needed to manage the cluster constantly; and 2) if a cluster is in the saturated mode and there is no other cluster in the shared mode, then requests of the class of the saturated cluster will be rejected. In order to generate a better availability of resources of the system, agent based architecture was developed to monitor the clusters load. This new architecture is presented in section 4.

4. THE DARC ARCHITECTURE: DYNAMIC RECONFIGURATION

In this section, we describe the *Dynamic Architecture for Reconfiguration of Web servers Clusters* (DARC) proposed in this paper. The main objective is to perform a self-reconfiguration of the resources in a web server to help the administrator, minimizing his stress in moments of work overload, using a multi-agent system. The DARC architecture enables the WS-DSAC to continuously meet the requests, thanks to the reconfiguration of the resources or threshold updates. A multi-agent system is a natural approach to perform a dynamic management of resources in a distributed way.

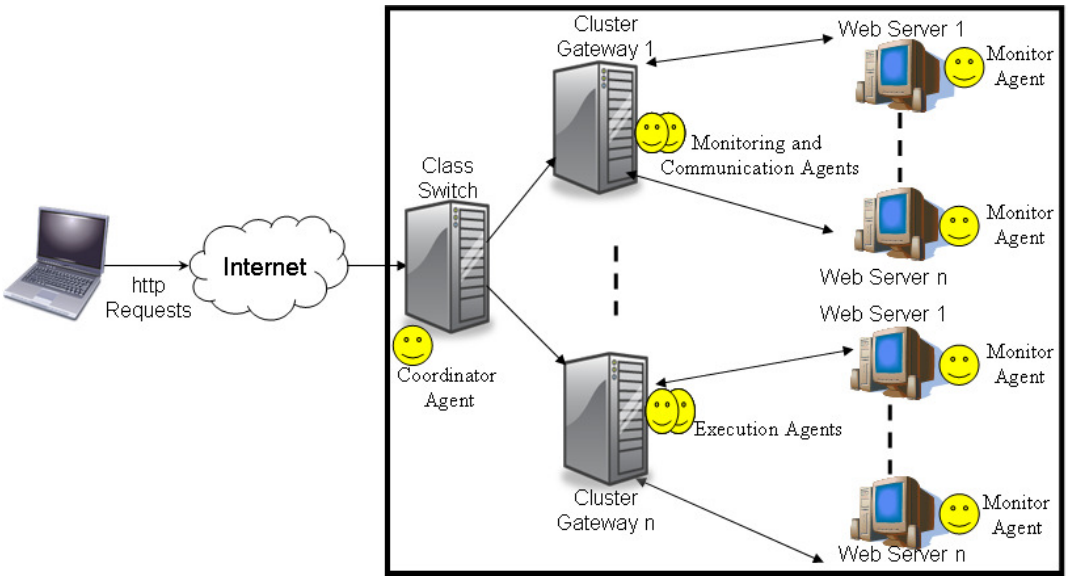

Figure 2. Agents in the DARC architecture

Thus, the agents are able to adapt the cluster to changing request patterns or environment factors (e.g., new servers can be added to the cluster -or removed from the cluster- easily). The agents reconfigure the system resources by interacting with the basic WS-DSAC platform described in the previous section. By interacting with the environment, these agents are able to learn from past experiences to modify, if needed, the distribution of the web server nodes in each cluster, allocating a host to a cluster that is overloaded and the classes' thresholds (R_{max} and R_{ac}) appropriately. Figure 2 shows the placement of the DARC agents on the different elements in a cluster.

Initially the cluster administrator defines different classes of services as well as the characteristics of each class (Figure 3).

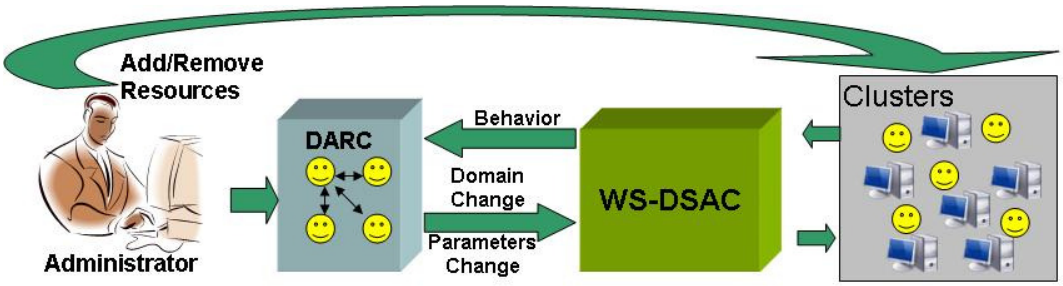

Figure 3. Dynamic capacity management

He/she also defines the clusters and hosts that are associated to such classes. While in operation, those features may change, requiring the intervention of the administrator to re-dimension the system. This work proposes an intelligent module that will learn how to interact with the framework, helping the administrator's work. The use of this architecture avoids the intervention and monitoring of the cluster administrator. Instead, the agents learn directly from the way its cluster operates and update the distribution of the web server nodes in each cluster and the classes' thresholds when is necessary, minimizing the probability of requests being rejected.

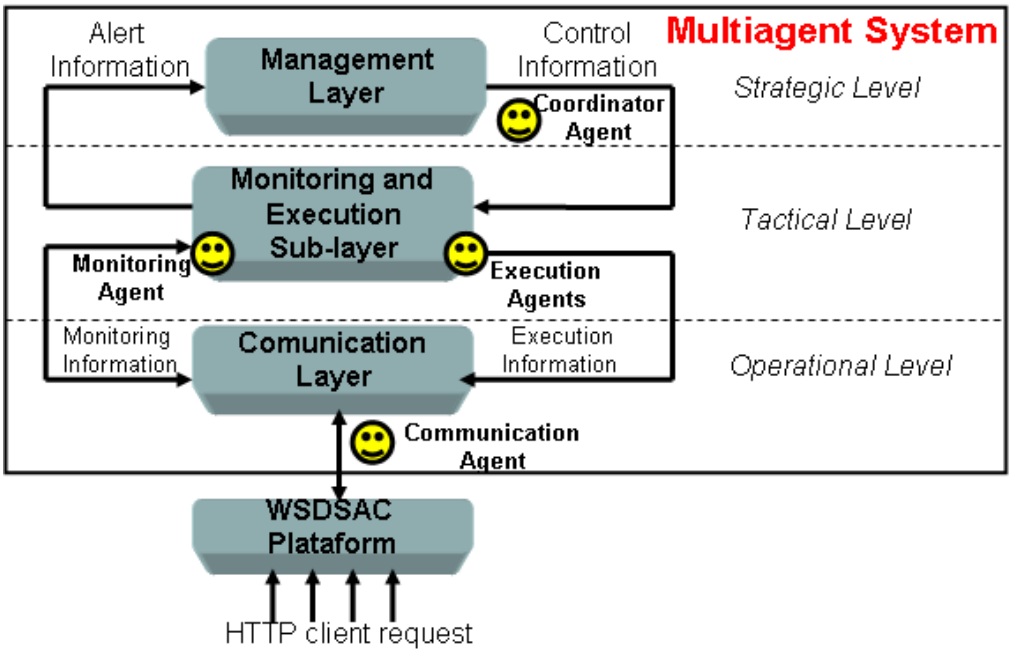

Figure 4. Levels and layers in DARC

Figure 4 presents the different levels of the DARC architecture existing within a cluster. The *Strategic Level* corresponds to the *Management Layer,* which manages the interactions among the various agents in the proposed architecture. The *Tactical Level* holds both the *Monitoring Sublayer* and the *Execution Sublayer*. Finally, the *Operational Level* corresponds to the *Communication Layer*, which is responsible for the communication with the WS-DSAC. As shown in figure 4, the multi-agent architecture proposed in DARC is composed of different types of agents:

- A *Communication Agent* requests the WS-DSAC platform about the value of the variable ρ_{ki} (see Section 3) and communicates this value to the Monitoring Agent (explained below).

- A *Monitoring Agent* keeps collecting information of the state of each server during specific time intervals. The monitoring rate is adjusted dynamically depending on the CPU load.

- A *Coordinator Agent* manages the interactions among several agents and receives alerts from the monitoring agents.

- Three *Execution Agents* can be distinguished according to their roles. Thus, the *MaximumLoad Agent* allocates the less loaded host to the cluster that is saturating. The *DynamicThreshold Agent* is designed to update clusters' thresholds before it saturates, acting only in case the action of the MaximumLoad Agent is not enough to prevent the cluster saturation. Finally, the *UpdateManager Agent* is designed to monitor the suitability of the update made by DynamicThreshold Agent. We will explain these mechanisms in more detail in the following.

The Execution Agents receive control information from its Coordinator Agents to set appropriate thresholds by taking into account the whole system. To avoid wrong decisions when overload spikes occur, we used a variable that stores the average load of the cluster (*medLoad*). In addition, we use the variables γ_{low} and γ_{high} (boundary parameters of the cluster load to work in exclusive mode) and δ_1 (update parameter of the cluster load to work in exclusive mode) to control the strategy adopted by the DARC architecture. The basic aspects of this mechanism are summarized in Figure 5.

If there exists a cluster in shared mode then
  do nothing
else if (medLoad >= γ_{low} * R_{ac} and medLoad < γ_{high} * R_{ac}) then
  the **MaximumLoad Agent** of the cluster that is overloading allocates the less load host of the system to that cluster
else if (medLoad >= γ_{high} * R_{ac}) then
  the **DynamicThreshold Agent** of the cluster in saturated mode increases the threshold of that cluster by δ_1.

Figure 5. Basic mechanism for DARC

Additionally, the following must be considered:

- There are situations that could compromise the performance of the self-reconfiguration approach. Thus, for example, it could happen that the DynamicThreshold Agent modifies the thresholds constantly, due to continuous alternate periods of overload spikes and periods of low overloading. The UpdateManager agent will make sure that the threshold adjustments are appropriate, by learning from past behaviors. Thus, if it detects several threshold updates in a

short period of time, it will increase the thresholds for its cluster (C1) by δ_1 and decrease the thresholds from the less loaded cluster (C2) by δ_1.

- In order to avoid overloading a server, the overall increase due to threshold updates cannot exceed 50% of the thresholds' initial values; that is, we assume that accepting requests when the load is above 1.5 * R_{ac} would result in overloading.
- When the cluster load stabilizes (i.e., when the cluster switches back to shared mode) the thresholds will be re-initialized.

Appropriate values of the parameters γ_{low}, γ_{high}, and δ_1, were chosen experimentally. Specifically, we set γ_{low} and γ_{high} to 80% and 90% of the cluster load to work in exclusive mode, respectively, and the value of δ_1 to 10% of the cluster load to work in exclusive mode. These values provided good results in a variety of experiments.

5. MODELING AND ANALYZING OF THE ARCHITECTURE IN COLORED PETRI NETS

5.1. Modeling

This section presents the modeling of DARC described in section 4 and illustrated in Figure 4. The modeling was done in Colored Petri Nets (CPN) [12] using the CPNTools. CPN is an adequate tool for modeling, simulating and analyzing of discrete event dynamics systems, among which the architecture proposal fits.

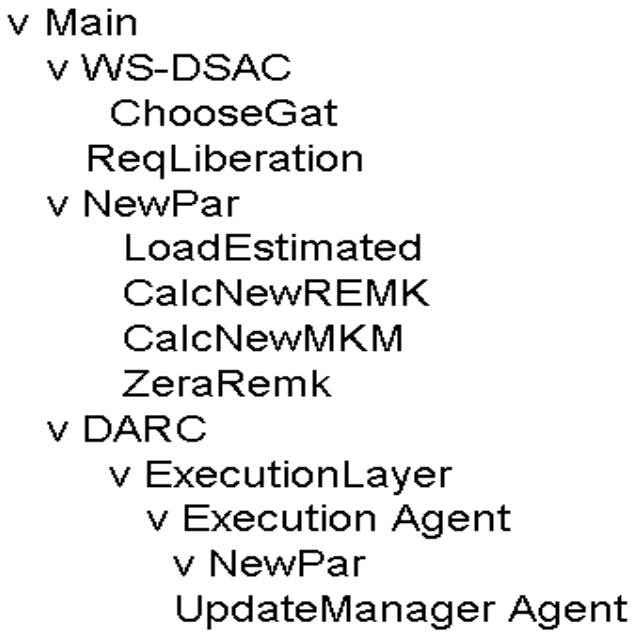

Figure 6. Hierarchy of Pages of the Modeling in Petri Nets

In Figure 6, the hierarchical view of pages and sub-pages of the model (represented by the substitution transitions - duplicate rectangles in the figures). The Main page presents the general operation of the WS-DSAC/DARC architectures detailed in Figure 2, and through it runs the others sub-pages: WS-DSAC, Liberation, NewParameters and DARC. The WS-DSAC sub-page performs the load balancing and it is based on differentiated services. This sub-page is composed of another sub-page (ChooseGat), where the load changes are modeled from a server. The Liberation sub-page performs the releasing of http requests after these requests are met. The

NewParameters sub-page has the function to updating the parameters used by the WS-DSAC platform, such as the calculation of new estimated loads of the servers belonging to clusters. Finally, the DARC sub-page presents the message exchange among the agents and their actions.

The modeling was done in two stages: the first based on the model of the WS-DSAC platform (I) and in the second the inclusion of the DARC architecture (II).

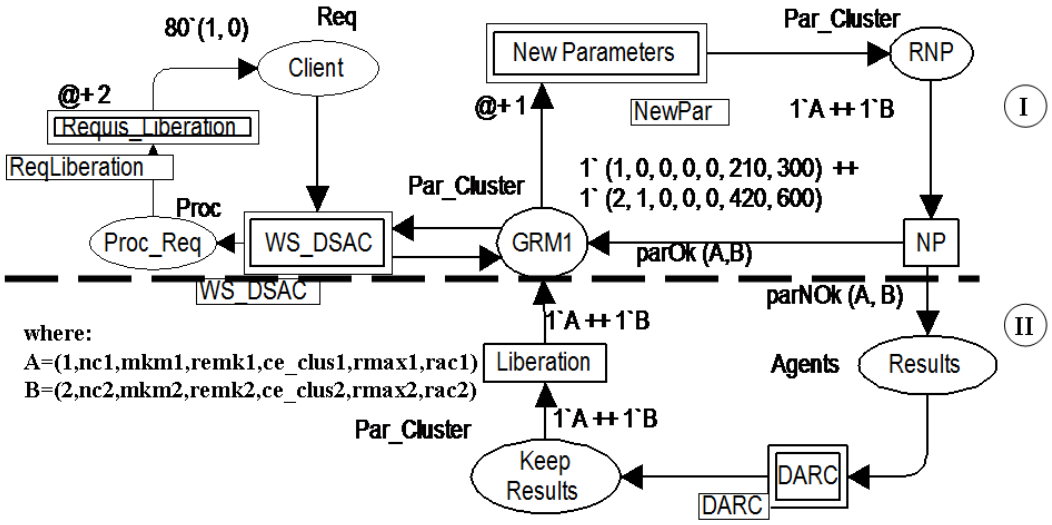

Figure 7. The Main Page of the Modeling in Petri Nets

In Figure 7, when a request arrives, the WS-DSAC platform is executed. The request and the load of the answering server are released. At pre-defined time intervals, the NewParameters sub-page is enabled and this sub-page performs the update of the parameters used by the WS-DSAC platform. After each update a check is performed by the *Monitoring Agent*. If the check is positive, then the DARC substitution transition (Figure 8) is executed and then the *Monitoring Agent* will send an alert to the *Coordinator Agent* that activates the *Execution Agents* (Execution Layer transition). Otherwise, the updated parameters of the WS-DSAC return to be used by the network and the processing of the requests continues.

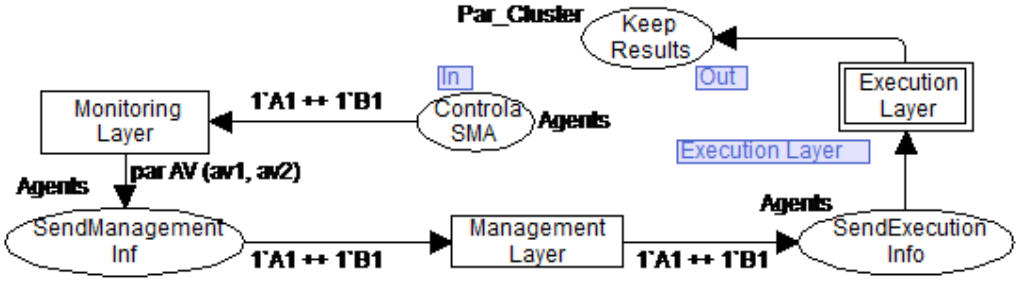

Figure 8. DARC Page of the Modeling in Petri Nets

From the model presented, several simulations were performed with different environments of operation of system for validation of DARC architecture. These simulations were performed using the CPNTools.

5.2. Simulation of the Model and Analyzing of the Results

The simulations were performed in two stages: first, we evaluate the WS-DSAC platform, without the DARC architecture, for dynamic reconfiguration, i.e., only the WS-DSAC platform

was simulated (CPN model I in Figure 7). In the second stage, the DARC architecture was added to WS-DSAC, and CPN model II was added to the Figure 7.

For the simulations, the classes 0 and 1 have been defined in the platform whose parameters are described in section 3. We used architecture with two clusters (clusters 0 and 1); each one had two servers. The cluster 0 was associated to a native class 0 and the cluster 1 to a class 1. So in the simulation, using DARC, the increase of resources in a cluster, such as the removal of a server in a cluster and its addition to the other cluster, is represented by the increasing of the parameters of a class of service in the same proportion of the decreasing of the parameters of the other. The values of variables R_{ac} and R_{max} are 300 and 210 respectively for cluster 0. For cluster 1, the values are respectively 600 and 420. These initial values were determined through an extensive experimental evaluation performed within the context of WS-DSAC [3]: They lead to a good performance in a variety of scenarios.

The simulation results are compared and presented in the following sections.

5.3. Critical Moments

Several experiments were performed using the CPNTools to evaluate our proposal. In each experiment, a client sends every second, a request belonging to class"0" while another client sends the same amount of requests belonging to the class "1". First, we evaluate the WS-DSAC platform, without the DARC architecture. We will call this approach *no-DARC1-sim*. For dynamic reconfiguration we evaluate the WS-DSAC with the DARC architecture. We will call this approach *DARC1-sim*.

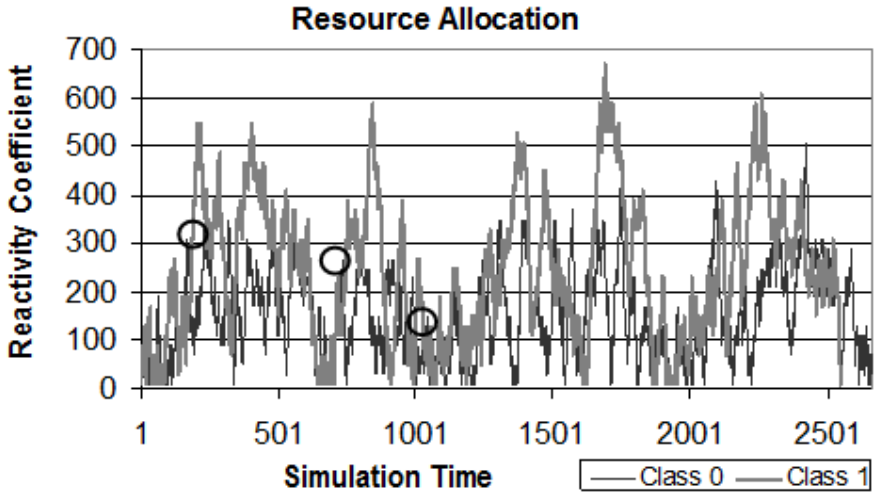

Figure 9. no-DARC1-sim

Figure 9 shows the distribution of load between the two clusters along time, using no-DARC1-sim. The simulations showed the first rejection of class 0 (RC > 300) at t=168 and, after that, other rejections are repeated during the simulation. This moment is signaled with the first circle from left to right. The simulations also showed the behavior of clusters, as in example of at t=785. The cluster 0 is overloaded (RC=210), changing their work mode to the *exclusive mode*. The cluster 1 will meet the requests in *shared mode*, RC<420. This load distribution does not prevent the system rejects requests because the clusters sometimes exceeded their thresholds. This situation is shown in Figure 9 by the second circle. The third circle presents the clusters in *shared mode*.

On the other hand, we evaluate the benefits of dynamic reconfiguration with the DARC architecture in Figure 10. The distribution of load between the two clusters along time shown in Figure 10, using the DARC1-sim approach with the same amounts of requests belonging to the two classes, the cluster 1 did not reach the *saturated mode* and it remained with the RC<600. To add, the cluster 0 exceeded less often its limit. Comparing with the results of simulation in Figure 9, the cluster 1 exceeds this limit causing a higher rate of rejection of requests, and the cluster 0 also obtained its maximum point (RC=510) against (RC=450) (see circle in Figure 10).

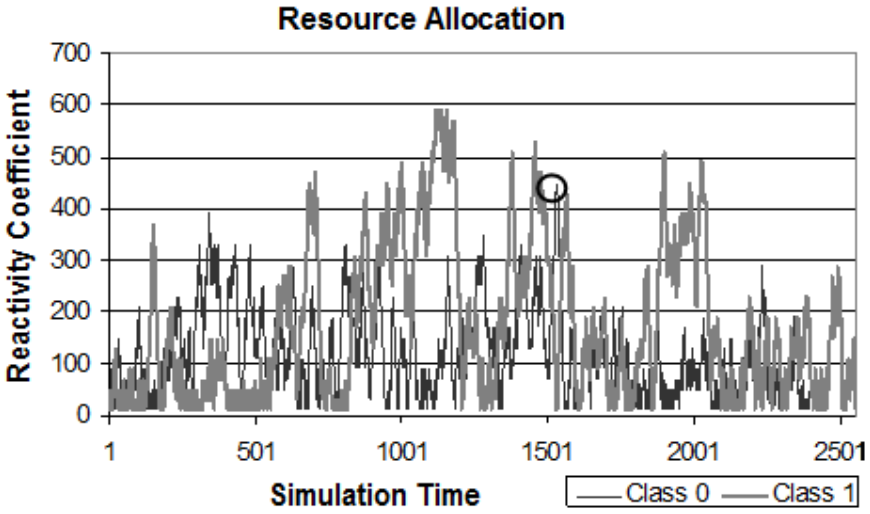

Figure 10. DARC1-sim

We conclude that the use of the DARC caused a dynamic reconfiguration of resources between the two clusters and the consequent decrease in saturation of clusters. In both cases (*no-DARC1-sim* and *DARC1-sim*), several simulations were made with the purpose of calculating the rejection rate, and it was observed that in the first case, the average rejection rate of requests was 3.93 % while the second was 0.04%, showing a better utilization of system resources.

5.4. Saturation Moments

In the following simulations were sent only requests belonging to the class 0 (Figures 11, 12, 13 and 14 show the results of simulations). Thus, the cluster 0 saturates more quickly because it has the lowest thresholds. The graph in Figure 11 refers to the WS-DSAC platform, without the DARC architecture, unlike the Figures 12, 13 and 14 where the DARC was applied. It is important to mention that the percentage of monitored load of clusters by agents was modified at several levels, being 80% in the first case (Figure 12), 85% in the second (Figure 13) and 90% in the last (Figure 14). In addition, 10% of resources were moved from cluster 1 to cluster 0, when needed, in all three cases. Referring to the simulation of Figure 12 as an example, it means that when the load of cluster 0 reached 80% of its load limit, 10% of the resources of cluster 1 were moved to him.

Figure 11 shows the distribution of load between the two clusters along time of the WS-DSAC, without DARC. We will call this approach *no-DARC2-sim*. Although this distribution occurs, there is also a considerable rejection of requests due to the saturation of cluster 0 (RC> 300). On the other hand, Figure 12 shows the distribution of load between the two clusters along time of the WS-DSAC with DARC. We will call this approach *DARC2-sim*. This simulation shows that the cluster 1 has not changed its work mode to saturate, though its resources are allocated to cluster 0. This can be verified by checking the points of maximum load of cluster 1 using the

two referred simulations. The Figure 11 shows that cluster 1 reaches a maximum load (RC = 530) at t= 1254, and in Figure 12 its peak load (RC=350) occurred at t=656 and the cluster worked until the end of the simulation below that value. Two moments of saturation of cluster 0 are represented by circles in Figures 11 and 12.

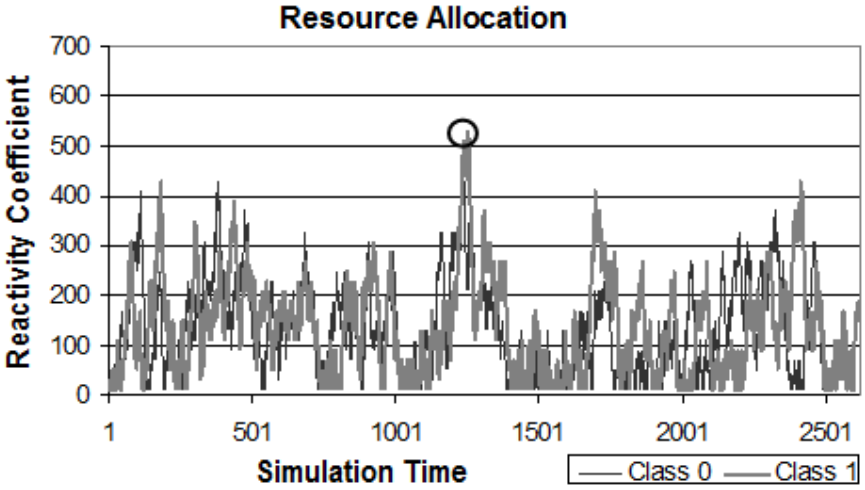

Figure 11. no-DARC2-sim

As in the two previous situations (*no-DARC1-sim* and *DARC1-sim*), several simulations were made with the purpose of calculating the rejection rate. They showed that in the first case, the average rejection rate of requests was 9.8% (Figure 11) while the second was 0% (Figure 12). Thus, the cluster 1 has not changed its work mode to saturate, though its resources are allocated to cluster 0. Consequently, the system met all requests sent improving the use of resources.

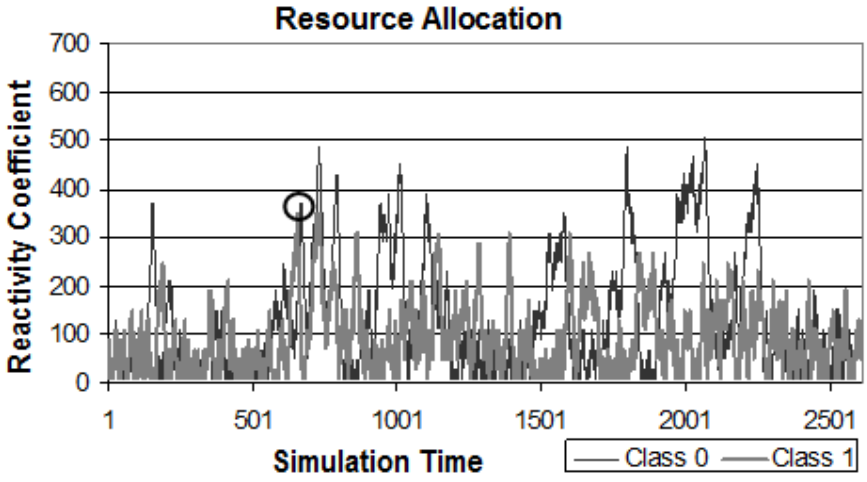

Figure 12. DARC2-sim

In the simulation on Figure 13 the *monitoring agent* notifies the *coordinator agent* when the load of the cluster 0 reaches 85% of its limit and Figure 14 when it reaches 90%. We will call the first approach *DARC3-sim* and the second approach *DARC4-sim*.

The rejection rate observed in the network was 0.02% on Figure 13 and 0.05% on Figure 14, indicating that the monitoring percentage of the threshold of the cluster is released, the rejection

rate also increases. It indicates that if the approach to be taken by agents is made over time, it is possible to get a better performance of the system.

The comparison of the behavior of the approaches analyzed presented in Figures 12, 13 and 14 are described below:

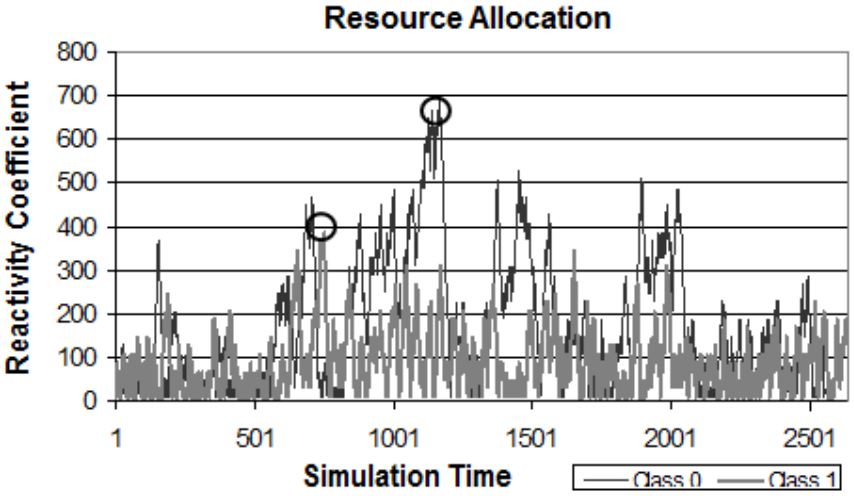

Figure 13. DARC3-sim

- Figure 12 shows that there are 13 load peaks ($RC>=R_{max}$) for the cluster 0, against 15 by the same cluster in Figure 13, and 17 load peaks in Figure 14. This indicates that at the moment that the reconfiguration is made with the monitoring percentage of 80% of the cluster threshold, the system requires fewer adjustments. On the other hand, when the percentage of monitoring the load is greater (85% and 90%) the rejections of requests are greater;

- Figure 12 shows that the load of cluster 1 was lower in most time with $RC<300$ and it was obtained less response time;

- Figure 13 shows that the clusters behaved more irregularly than in Figure 12, reaching load values close to the limits set by the administrator. Indeed, the simulations showed at t=1164 to cluster 0, and t=748 for cluster 1. The circles in the figure indicate the location of these moments;

- Figure 14 shows that the cluster 1 has reached a value of load that exceeded the limit at t=555, signaled by the circle. As shown, in several other moments of the simulation, values were very high, even more than in other simulations (Figures 12 and 13).

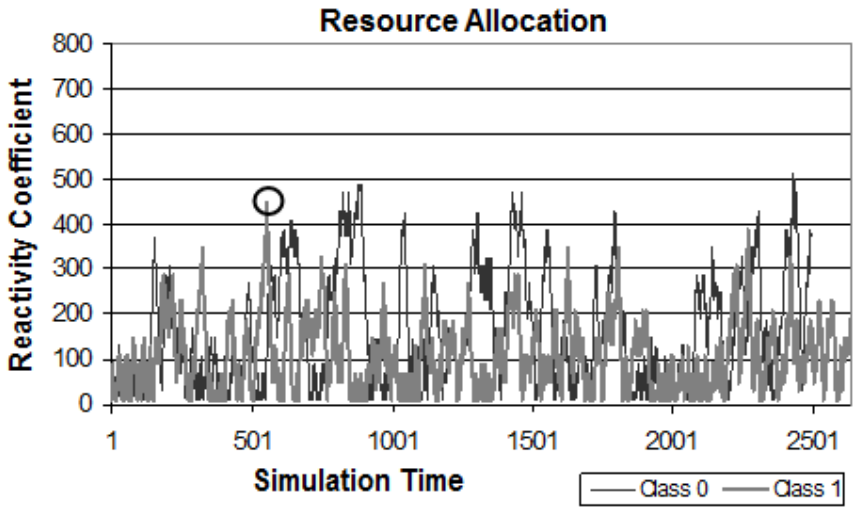

Figure 14. DARC4-sim

With the results presented above, we can conclude that among all the simulations presented, the one that best behaved was the simulation that made the reallocation of resources to monitoring 80% of the load of cluster 0 (Figure 12), which had the most consistent proposed objectives of this work. In addition, we can conclude that the delay in taking actions for reconfiguration of resources in clusters causes an overload on their servers and their consequent rejection of requests. Thus, the incorporation of DARC to WS-DSAC makes the best dynamic reconfiguration of resources, reducing the number of rejections, and therefore significantly improving the use of system resources and reducing the rejection rate of requests.

5.5. Performance Evaluation

Figure 15 presents the response time given by the performance model of WS-DSAC without DARC and WS-DSAC/DARC.

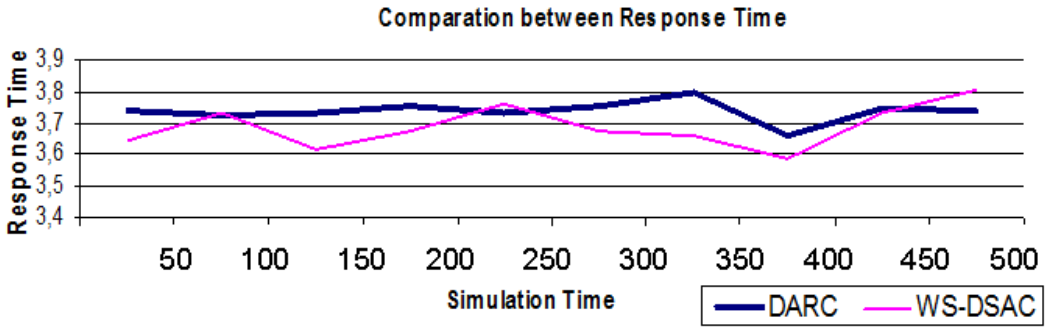

Figure 15. Comparison of response times

We can observe in figure 15 that, in the simulation, the response time of the DARC architecture is a little larger than the WS-DSAC platform. However, the DARC architecture meets more requests and the rejection rate is lower, which proves the effectiveness of the solution. Figure 16 presents the rejected rate given by the performance model of WS-DSAC/DARC. The rejection rate of the WS-DSAC platform is 6%. However, the DARC architecture is 0.02 %.

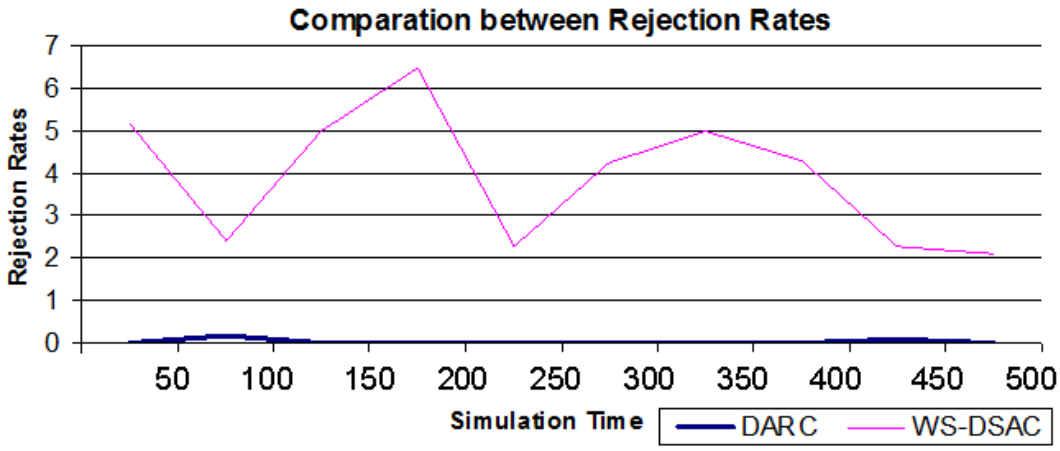

Figure 16. Comparison between rejection rates

6. IMPLEMENTATION AND EVALUATION OF DARC ARCHITECTURE TEXT

To evaluate our proposal, we have performed several experiments in a real environment. Each experiment is repeated several times and average results are reported, checking that the averages are significant. As Figure 17 shows, a heterogeneous scenario (with different operating systems and hardware configurations) and two clusters (*Cluster 0* and *Cluster 1*) has been considered for experimental evaluation:

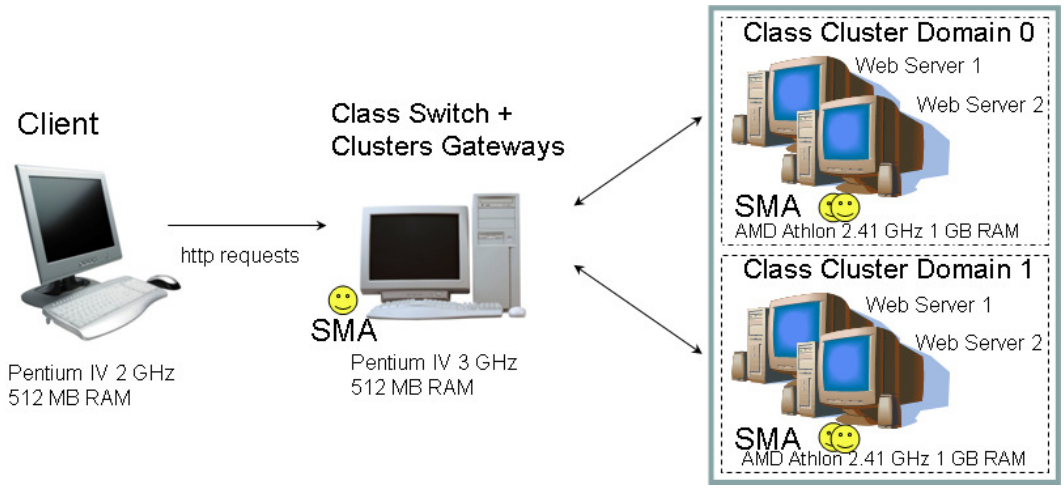

Figure 17. Hardware for the experiments

- Web server nodes 1, 2, 3 and 4 are generic PCs, with an AMD Athlon 64 Processor 3800, 2.41 GHz CPU, 1 GB RAM, and a 100 Mb full duplex Ethernet NIC. In these nodes, the *Apache Tomcat 5* Servlet/JSP Container (http:\\tomcat.apache.org) is installed.
- The Class Switch and the Cluster Gateway are generic PCs with an Intel Pentium IV, 3.06 GHz CPU, 512 MB RAM, and a 100 Mb full duplex Ethernet NIC.
- The client node is a generic PC, with an Intel Pentium IV, 2 GHz CPU, 512 MB RAM, and a 100 Mb full duplex Ethernet NIC. We have used *Webserver Stress Tools 7 Professional Edition* (http://www.paessler.com/webstress) to emulate the sending of HTTP requests: The Class Switch domain receives one request every second.

For the experiments presented in this section, we consider the platform described in Figure 17.

First, we evaluate the WS-DSAC Platform, without the DARC architecture, for dynamic reconfiguration. We will call this approach *no-DARC*. Figure 18 shows the distribution of load between the two clusters along time, for a scenario with an initial low load that increases considerably in the final moments of the simulation. Although Cluster 0 exceeds the value of R_{ac}=300 at time instants 120ms and 270ms, no request was rejected because Cluster 1 was in shared mode. However, at time instants of 180ms, 430ms, and 830ms, Cluster 0 exceeds its threshold R_{ac} and Cluster 1 was in exclusive mode, leading to rejections of requests of the class assigned to Cluster 0. Similarly, Cluster 1 exceeds the value of R_{ac}=600 at the end of the simulation while Cluster 0 is in exclusive mode, leading to rejections of the requests for Cluster 1.

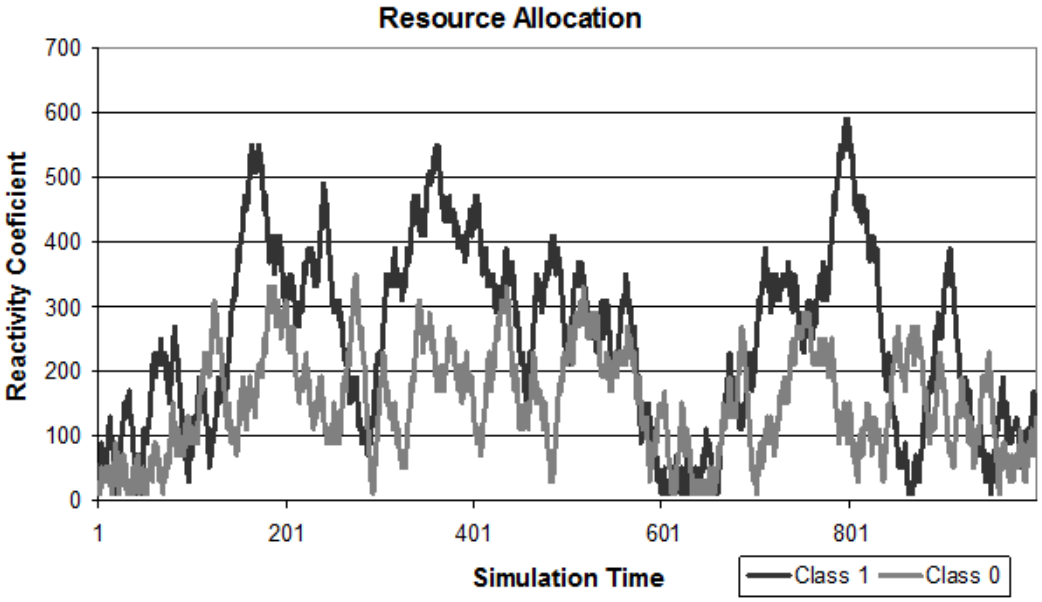

Figure 18. no-DARC

Second, we evaluate the WS-DSAC Platform, with the DARC architecture, for dynamic reconfiguration but without using the DynamicThreshold Agent or any learning mechanism. We will call this approach *DARC-1*. In Figure 19, Cluster 0 exceeds the value of R_{ac} only at time instant 430ms; as Cluster 1 is in exclusive mode at that time, then the requests of the class assigned to Cluster 0 will be rejected. However, regarding the previous experiment where DARC is not used (whose results are shown in Figure 18), the amount of requests rejected decreases.

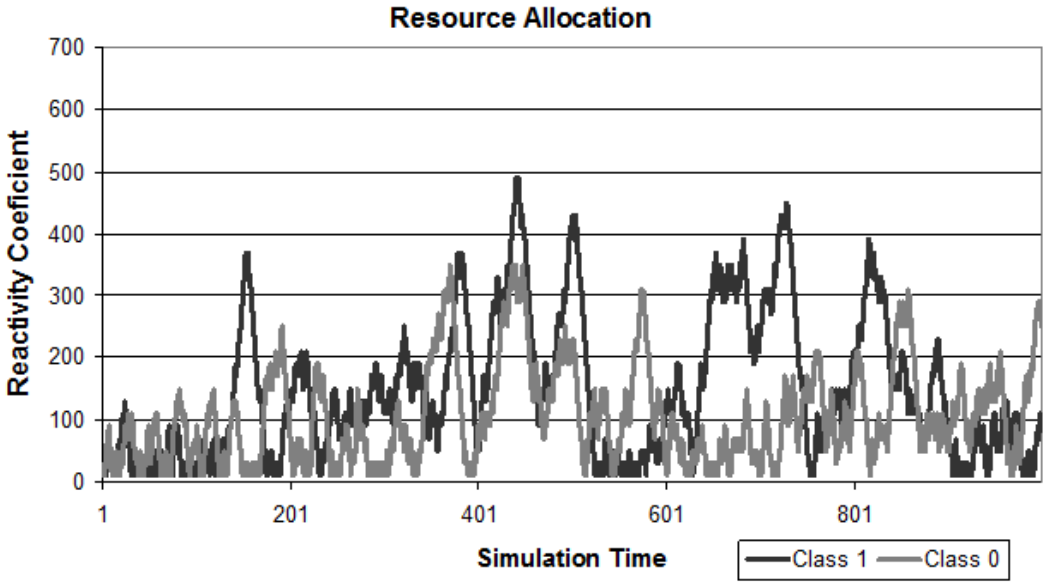

Figure 19. DARC-1

Third, we evaluate DARC-1 with the addition of the DynamicThreshold agent to improve fairness in the use of resources. We will call this strategy *DARC-2*. The results are shown in Figure 20. At time instants 390ms and 930ms, Cluster 0 exceeded the threshold and it was not possible to allocate any host from Cluster 1 (that was in exclusive mode) to Cluster 0. To solve this problem, the DynamicThreshold agent updated the cluster's threshold sometimes (3-5 times in the different repetitions of the experiments). As mentioned in Section 4, if ($medLoad \geq \gamma_{high} * R_{ac}$) then the **DynamicThreshold Agent** of the cluster in saturated mode increases the thresholds of that cluster by δ_1.

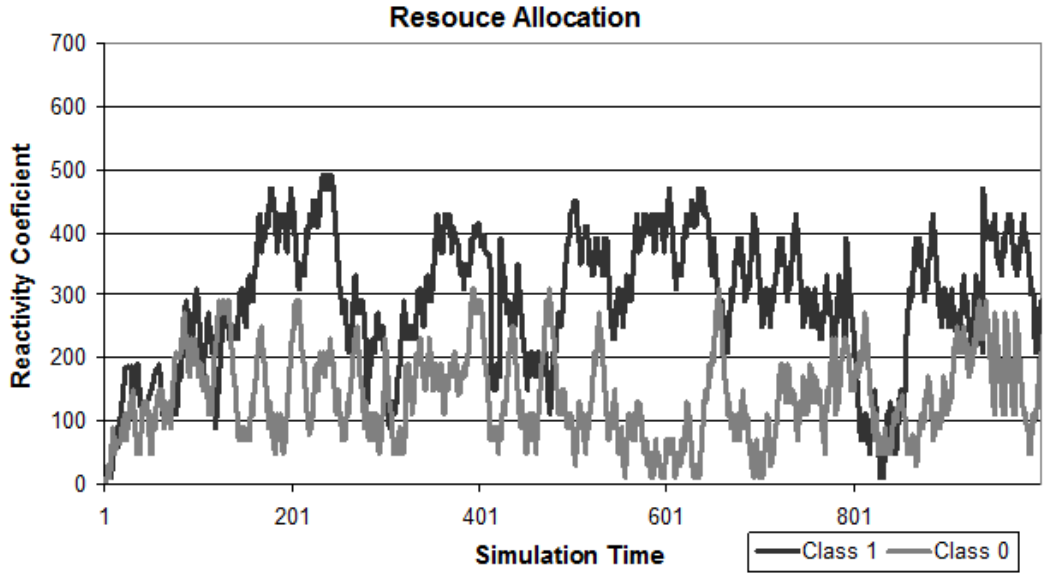

Figure 20. DARC-2

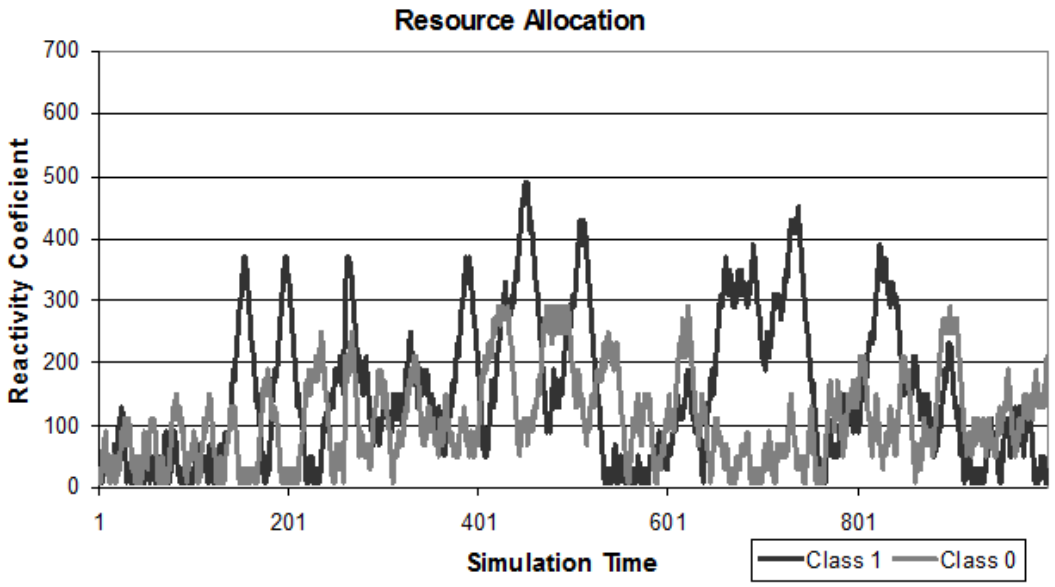

Figure 21. DARC-3

Finally, we perform an experiment with our full-fledge self-reconfiguration proposal (*DARC-3*), where an UpdateManager agent (that learns from the past history) is added. The results are shown in Figure 21. In this experiment, there were no peak saturations for the Clusters 0 and 1. With the utilization of the DARC architecture we can resolve the main problems of the WS-DSAC platform presented to meet the requests. Therefore, we reduce the peak saturation of the system and as consequence the rejected requests.

7. CONCLUSIONS

This paper has presented DARC, a dynamic architecture for self-reconfiguration of clusters of web servers. Our approach benefits from the use of agents to learn from the environment and adjust automatically the behavior of the system to make a better use of the available resources. With this approach, it's possible to help the administrator, minimizing your stress in moments of work overload. We performed the conception, specification, modeling in Colored Petri nets and implementation in Java of this architecture.

Finally, we performed experiments in different situations and we saw that the DARC architecture decreased the rejection rates and improved the availability of resources ("fairness"). Therefore, with the experiments we can prove that when the monitoring percentage of the load increased, the rejection rate of requests also increased.

It is important to emphasize that the main advantages of this architecture is that we can optimize the allocation of the resources from one cluster to another so that all requests are met. For the initial parameters used in our experiments, only 10% of the resources of a cluster were redirected to the other cluster, so that the rejection rate decreased and reached 0.

As for future work, we plan on analyzing the possibility of applying this proposal (probably with some extensions) in a different domain (not in the context of web servers).

REFERENCES

[1] Ayyasamy S.; Sivanandam S.N. A Cluster Based Replication Architecture for Load Balancing in Peer-to-Peer Content Distribution. In: International Journal of Computer Networks & Communications (IJCNC). v. 2, n. 5, p. 158-172, September 2010

[2] M. Wooldridge, (2002) An Introduction to MultiAgent Systems, John Wiley and Sons Ltd.

[3] A. Serra, D. Gati, G. Barroso, R. Ramos, and J. Boudy, (2005) "Assuring QoS diferentiation and load balancing on web servers clusters", *International Conference on Control Aplications*, pp. 885-890.

[4] R. Olejnik, A. Bouchi, and B. Toursel, (2002) "An object observation for a java adaptative distributed application platform", *International Conference on Parallel Computing in Electrica Engineering*, pp. 171-176.

[5] J. Wang, Y. Ren, D. Zheng, and Q. Wu, (2007) "Agent based load balancing middleware for service- oriented applications", *International Conference on Computational Science, Part II*, pp. 974-977.

[6] P. Herrero, J. L. Bosque, M. Salvadores, and M. S. Perez, (2007) "An agents-based cooperative awareness model to cover load balancing delivery in grid environments", *Lecture Notes in Computer Science*, pp. 64-74.

[7] N. Nehra and R. B. Patel, (2007) "Towards dynamic load balancing in heterogeneous cluster using mobile agent", *International Conference on Computational Intelligence and Multimedia Applications*, pp. 15-21, December.

[8] N. Nehra, R. B. Patel, and V. K. Bhat, (2006) "A multi-agent system for distributed dynamic load balancing on cluster", *International Conference on Advanced Computing and Communications*, pp. 135-138.

[9] Z. H. Zhang, D. Meng, J. F. Zhan, L. Wang, L. P. Wu, and W. Huang, (2006) "Easy and reliable cluster management: The self-management experience of Fire Phoenix", *International Parallel and Distributed Processing Symposium*, p. 8pp..

[10] H. Sung, B. Choi, H. Kim, J. Song, S. Han, C. W. Ang, W. C. Cheng, and K. S. Wong, (2007) "Dynamic clustering model for high service availability", *International Symposium on Autonomous Decentralized Systems*, pp. 311-317.

[11] C. Adam and R. Stadler, (2005) "Adaptable server clusters with QoS objectives", *International Symposium on Integrated Network Management*, pp. 149-162.

[12] K. Jensen, L. Kristensen, L. Wells (2007) *Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems*. Springer, vol. 9.

Authors

Carla Katarina de Monteiro Marques is an assistant professor at the Computer Science Department of the University do Estado do Rio Grande do Norte, where she teaches computer networks. She is a Master Degree in Computer Science, PhD student in Federal University of Ceara, and works in reconfiguration system, Agent based Simulation and Coloured Petri Nets.

Isabel Cristina Régio de Oliveira is a Masters Student at the Engineering Teleinformatica Department of the Federal University of Ceara, where she works in Clusters Web based Simulation and Modelling, Agent based Simulation and Modelling, and Coloured Petri Nets

Antônio de Barros Serra is a professor at the Telematica Engineer Computer Science Department of the Federal Institute of Educ. Ciencia e Tecnologia, where he teaches computer networks. He is a PhD in Réseaux Avancés de Connaissances et Organisations pela Institut National des Télécommunications, and works in Cluster Computing.

Giovanni Cordeiro Barroso is an adjunct professor at the Physics Department of the Federal University of Ceara, where he teaches modeling hybrid systems. He is a Pós-PhD in Teleinformatica pelo Institut National des Télécommunications - INT, Evry-France and works in Petri nets, modeling and analysis, coloured petri nets, distributed systems and discrete event systems.