Performance analysis of a Dynamic Architecture for Reconfiguration of Web Servers Clusters

Carla Katarina M. Marques
Univ. of Estado R.G. do Norte, Brazil
carla.katarina@gmail.com

Sergio Ilarri and José Merseguer
Universidad de Zaragoza, Spain
{*silarri,jmerse*}*@unizar.es*

Giovanni C. Barroso
Univ. of Estado R.G. do Norte, Brazil
gcb@fisica.ufc.br

***Abstract*—The administration of clusters is an exhausting job. Particularly, allocating the resources of the clusters by hand can easily become unmanageable because the processing requirements can change very quickly in a dynamic environment such as the Internet. A solution to solve this problem is to use a dynamic architecture for self-reconfiguration of the clusters.**

In a previous work, we have proposed the DARC architecture, an agent-based architecture that can perform an automatic reconfiguration to adapt itself to the current needs. In this paper, we formally model our architecture using SPE techniques. These models are validated by comparing the analytical results with results obtained through experimental evaluation. The models obtained can thus be used to evaluate DARC in different environments without the hassle of a time-consuming experimental evaluation. For example, in this paper we have used our models to compare the strategy of load balancing without self-reconfiguration with an approach with self-reconfiguration. This paper also shows that the use of Generalized Stochastic Petri Nets (GSPN) is suitable to analyze complex performance problems in the dynamic reconfiguration domain.

***Keywords*-UML; software agents; software reconfiguration; performance.**

I. Introduction

In recent years, cluster computing technology has become a very attractive platform for low-cost super-computing, since it is easy to build and it offers interesting advantages such as high availability. Basically, it consists of several workstations interconnected through a high-speed network for information exchange and coordination among them.

However, the management of a cluster is an exhausting job, even for experienced cluster administrators. Particularly, allocating the resources of a cluster by hand can easily become unmanageable. Thus, the processing requirements can change very quickly in a dynamic environment such as the Internet. A solution to solve this problem is to use a dynamic architecture for reconfiguration of the resources in a cluster. To perform this dynamic reconfiguration in a cluster of web servers, we have previously proposed the *DARC* (*Dynamic Architecture for Reconfiguration of Web Servers Clusters*) architecture [1], which performs the self-reconfiguration using a multiagent system [2]. The agents of the architecture interact to modify the distribution of the web server nodes in each cluster, allocating/deallocating nodes to/from clusters as needed in order to satisfy the current requirements.

In this paper, we study the performance of DARC from an analytical perspective, by following the *SPE* (*Software Performance Engineering*) techniques [3] to formally model and analyze this architecture. The performance analysis of a software specification in a language such as *UML* (*Unified Modeling Language* [4]) can assist a design team in evaluating performance-sensitive design decisions and in making design trade-offs that involve performance. Annotations to the design based on the *UML Profile for Schedulability, Performance and Time* [5] are used to provide the necessary information for a performance model (such as the workload parameters) and many different kinds of performance techniques can then be applied [6]. Our analysis allows us to validate the experimental results previously obtained [1] and to compare the strategy of load balancing without self-reconfiguration with the approach with self-reconfiguration.

The structure of the rest of the paper is as follows. In Section II, we introduce some basic aspects of DARC. In Section III, we model the DARC architecture using UML. In Section IV, we describe and model the DARC architecture from a performance perspective, by annotating the previous models with performance information. In Section V, we validate and exploit the performance models of the DARC architecture to evaluate it analytically. In Section VI, we present the related work. Finally, in Section VII, we draw some conclusions.

II. Overview of DARC

To understand the rest of the paper, an overview of *DARC* (*Dynamic Architecture for Reconfiguration of Web servers Clusters*) is needed. In this section, we summarize the basic aspects of DARC, which was described in more detail in [1].

To perform the dynamic reconfiguration of resources, DARC interacts with a platform of web clusters. A cluster of web servers consists of three basic components: the *Class Switch*, the *Cluster Gateways*, and the *Web Servers*. Requests of web resources by clients are grouped in different *classes* (defined by the administrator) according to their priority. The Class Switch is responsible for the classification and admission of new client requests. It receives incoming HTTP requests, identifies the class of the request, verifies the current load of the *class cluster domain* (each cluster is associated with a class of QoS) and sends each request to the less loaded Cluster Gateway for that class. The corresponding Cluster Gateway sends this request to the less loaded server in the cluster. The cluster of web servers manages different operating modes; e.g., a web server may accept requests of a specific class (exclusive mode) or requests of several classes (shared mode). Switching from one mode to another is controlled by different thresholds: ρ_{ki} (the *reactivity coefficient*, which controls how the system will switch from one operating mode to another), $Rexcl$ (maximum value of ρ_{ki} to work in shared mode), $Rsat$ (maximum value of ρ_{ki} to work in exclusive mode), and *medLoad* (average load of the cluster); for more details, see [1].

The main advantage of the DARC architecture is that it performs a self-reconfiguration of the resources in a web server without the need of a cluster administrator. For this purpose, it uses a multiagent system, which is a natural approach to perform a dynamic management of resources in a distributed way. The agents observe the way in which their cluster operates and update the distribution of the web server nodes in each cluster (e.g., by allocating a host to a cluster that is overloaded) and the values of the aforementioned thresholds when necessary, minimizing the probability of requests being rejected due to the lack of resources in the cluster. As shown in Figure 1, the multiagent architecture proposed in DARC is composed of four different types of agents:

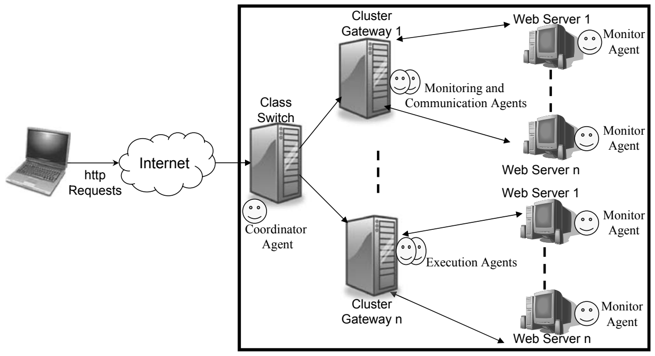


Figure 1. Agents in the DARC architecture.

- A *Communication Agent* requests the platform of web clusters about the value of the threshold ρ_{ki} and communicates it to the Monitoring Agent (explained below).
- A *Monitoring Agent* collects 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 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 the thresholds of a cluster before it saturates, acting only in case the action of the MaximumLoad Agent is not enough to prevent the overloading of the cluster. Finally, the *UpdateManager Agent* is designed to monitor the suitability of an update made by DynamicThreshold Agent.

III. Modeling of the DARC Architecture

The sequence diagram in Figure 2 models how the agents interact. The *Monitoring Agent* requests monitoring information about the load of the clusters to the *Communication Agent* of the clusters. To obtain the load information of the platform of web clusters, the *Communication Agent* requests this to the Cluster Gateway by using the *getLoadInf* method, and communicates this value to the *Monitoring Agent*. Then, the *Monitoring Agent* tests a condition to decide how to proceed. Specifically, if *medLoad*$_k \geq \gamma_{low} {*} Rsat_k$ and *medLoad*$_k < \gamma_{high} * Rsat_k$ are both true, then the *Execution Agent a1:MaximumLoad Agent* will execute; this agent allocates the less loaded host to the cluster that is saturating. Otherwise, if *medLoad*$_k \geq \gamma_{high} * Rsat_k$ is true, then the *Execution Agent a2:DynamicThreshold Agent* will execute. This agent is designed to update the thresholds of a cluster before it saturates, acting only in case the action of the *Execution Agent a3:MaximumLoad Agent* is not enough to prevent the saturation of the cluster. Thus, if many updates of thresholds occur in a short period of time then the *UpdateManager Agent* will execute and 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 (i.e., C2 will allocate own resources in the proportion of δ_1 to process requests for C1).

The *WSDSAC* (*Web Servers - Differentiated Services Admission Control*) platform [7] has been integrated as part of the DARC Architecture. This platform has the following three main objectives: 1) to balance the load among the different computers available, 2) to guarantee different QoS (*Quality of Service*) levels, and 3) to use the available resources in an effective way. The sequence diagram in Figure 3 models the basic aspects of the WSDSAC platform. First, the *Class Switch* receives the incoming HTTP requests, identifies the class of each request by using the *httpidentification* method, and sends it to the corresponding Cluster Gateway in charge of managing requests of that class. The *Cluster Gateway* verifies the current load of its clusters, by computing a variable *medLoad*, and returns the requested information. Then, the *Cluster Gateway* tests if the condition (*Load*$_k \leq Rexc_k$) is true. If so, then the request is sent to

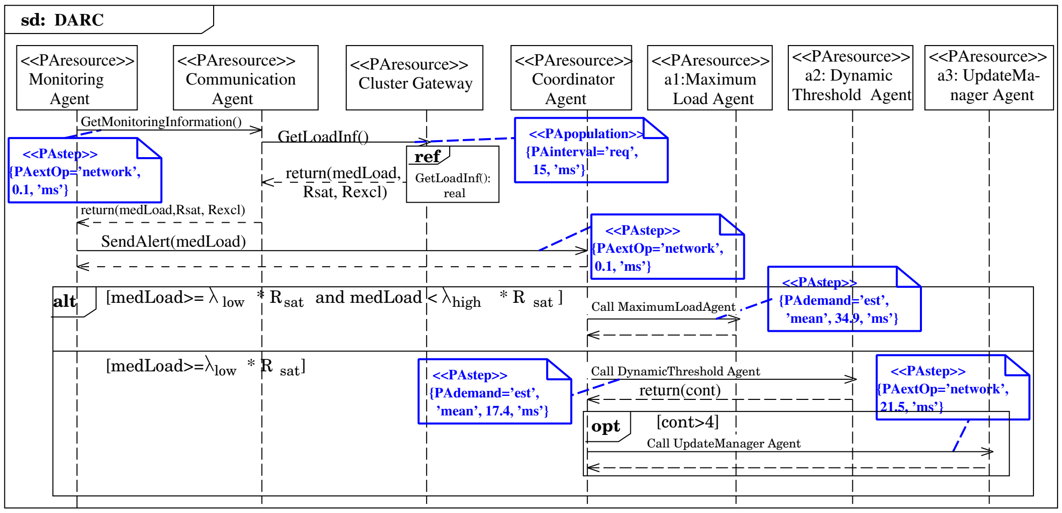


Figure 2. Sequence Diagram of DARC.

the less loaded cluster. Otherwise, if $Load_k > Rexc_k$ and $Load_k \leq Rsat_k$ then the request is sent to the native cluster for that class of request. If none of these two conditions hold, then the request is rejected.

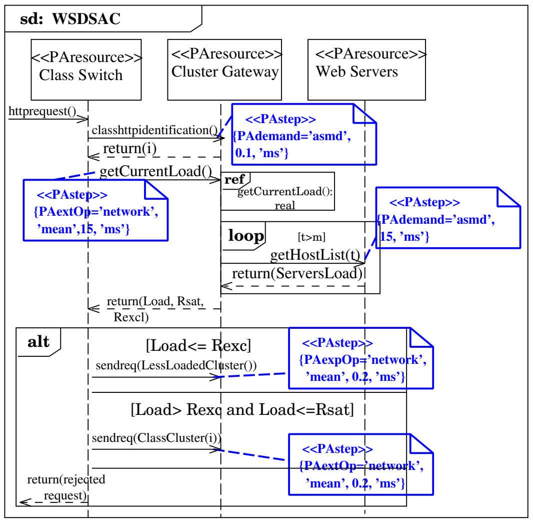


Figure 3. Sequence Diagram of WSDSAC.

To conclude this section, the modeling of the physical structure of the DARC architecture is provided in Figure 4 by using a UML *deployment diagram* (*DD*).

IV. Performance Models

The objective of this section is to get performance models, in terms of *Petri nets* (*PNs*), from the UML diagrams that model DARC. Hence, we aim to obtain performance models for DARC.

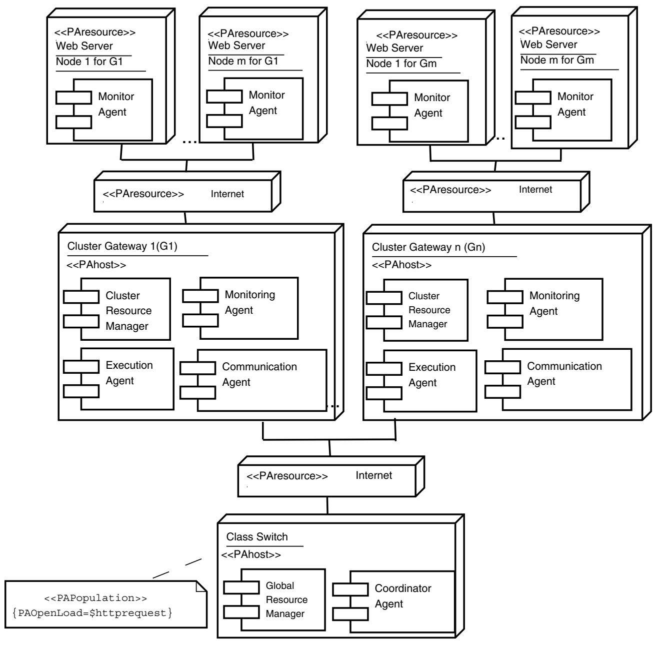


Figure 4. Deployment Diagram of the DARC Architecture.

A. UML-annotated models

We use the standard *UML Profile for Schedulability, Performance and Time Specification* (*UML-SPT* [5]) to introduce the input performance values of the DARC architecture and also the metrics that will characterize our performance analysis. SPT extends the UML standard by defining stereotypes and tags which can be applied to the model elements in the UML diagrams. See in Figures 3, 4 and 5 the stereotypes and tags used to introduce in DARC its intrinsic performance characteristics. These annotations appear attached as notes in the UML diagrams as SPT proposes. Consider that SPT

Operation	Average Execution Time (ms)
GetMonitoringInformation	0.1
GetCurrentLoad	15
GetLoadInf	15
SendAlert	0.1
CallMaximumLoadAgent	34.9
CallDynamicThresholdAgent	17.4
CallUpdateThresholdAgent	21.5
httprequest	0.01
classhttpidentification	0.1
GetHostList	15
sendreq	0.2

Table I
MEAN EXECUTION TIME FOR BASIC OPERATIONS.

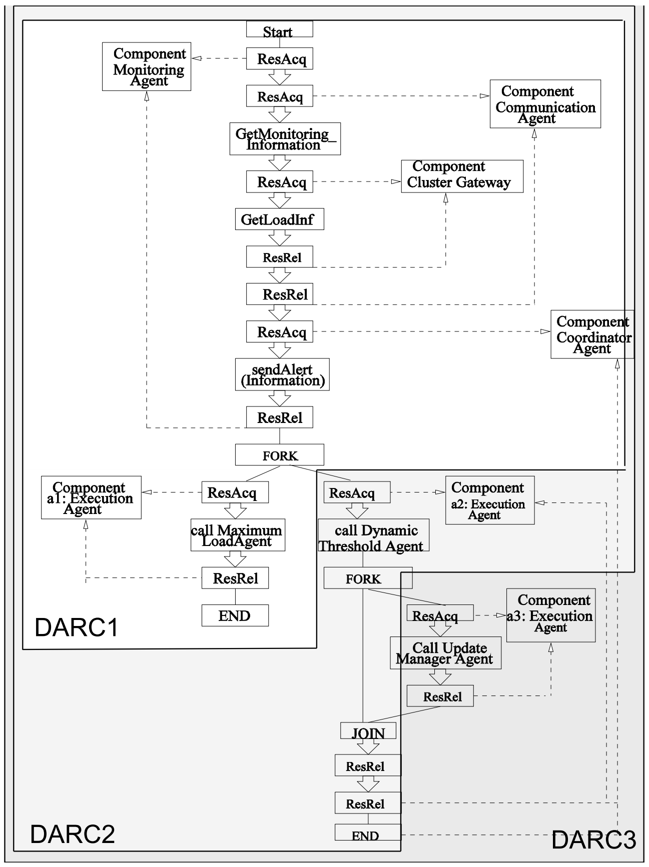
is mainly based on domain sub-models for resources and for performance, which indeed will be the basis of the CSM metamodel described in Section IV-B.

The input parameters in DARC will be the system load, the actions duration and the messages delays. They were collected by experimental tests using the DARC architecture [1]. Actions and messages are represented by the stereotype <<*PAstep*>>, where the *PAdemand* tag specifies the execution or delay time as an exponentially distributed random variable (see Table I). The system load is represented by the <<PApopulation>> stereotype, see Figure 3, and its tag *PAinterval* indicates that it is of the open kind and it requests every 15 milliseconds.

B. Core Scenario Model Diagrams

The *Core Scenario Model* (*CSM* [6]) is an intermediate performance model. The benefits of intermediate models are discussed in [6], and they basically bring the choice to be transformed into different formal models, such as PNs or queuing networks. The CSM is focused on describing performance *Scenarios*. A scenario, is a sequence of *Steps*, linked by *Connectors*. A step is a sequential piece of execution. Connectors can include branches, merges, and forks and joins. The scenario has a *Start* and an *End* points, where it begins and finishes. Start points are associated with at *Workload*, which defines arrivals and customers, and may be open or closed. There exist two kinds of *Resources*: *Active*, which execute steps, and *Passive*, which are acquired and released during scenarios by special *ResAcquire* and *ResRelease* steps. Steps are executed by (software) *Components* which are passive resources. A primitive step has a single host processor, which is connected through its component.

It is worth noticing that there exists an automatic translation from UML-SPT annotated models into CSM models [6], and also its associated tools. This means that the CSMs for DARC (Figures 5 and 7) could be easily obtained from their UML-SPT models (Figures 2 and 3, respectively). Although the automatic translation exists, it is straightforward to check some correspondences among models. For example, the *GetMonitoringInformation* message in Figure 2 is converted into an CSM *Step* in 5. The *ResAcquire* and *ResRelease* steps get hold and release the neccesary resources, that were deployed in the diagram in Figure 5.



Start
ResAcq
Component Monitoring Agent
ResAcq
Component Communication Agent
GetMonitoring_ Information
ResAcq
Component Cluster Gateway
GetLoadInf
ResRel
ResRel
ResAcq
Component Coordinator Agent
sendAlert (Information)
ResRel
FORK
Component a1: Execution Agent
ResAcq
call Maximum LoadAgent
ResRel
END
ResAcq
Component a2: Execution Agent
call Dynamic Threshold Agent
FORK
ResAcq
Component a3: Execution Agent
Call Update Manager Agent
ResRel
JOIN
ResRel
ResRel
END
DARC1
DARC2
DARC3

Figure 5. CSM for DARC-1, DARC-2 and DARC-3.

C. Petri Nets

Petri nets (PN) are a graphical and mathematical modeling tool for describing concurrent systems. We use a temporal extension, the class of *Generalized Stochastic Petri Nets* [8] (*GSPN*), which distinguishes three kind of transitions: immediate transitions; transitions with probabilities; and transitions with exponentially distributed random firings.

Fortunately, there also exists an automatic translation from CSM models into GSPN [9]. Figures 6 and 8 depict the GSPNs that we obtained from the CSMs in Figures 5 and 7.

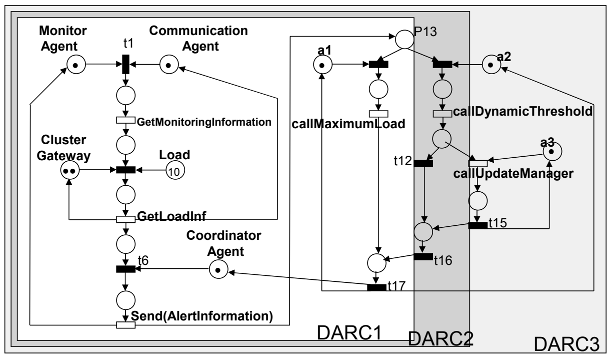


Figure 6. GSPN for DARC-1, DARC-2 and DARC-3.

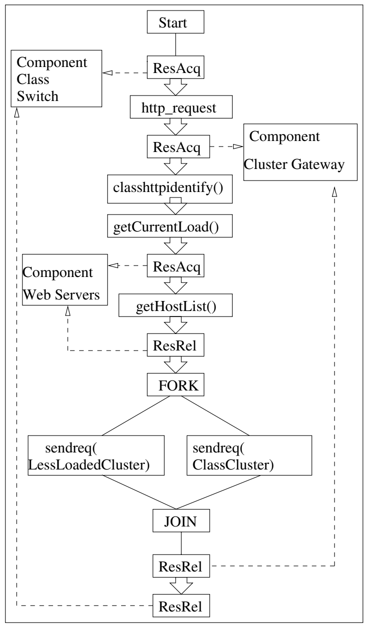


Figure 7. CSM WSDSAC.

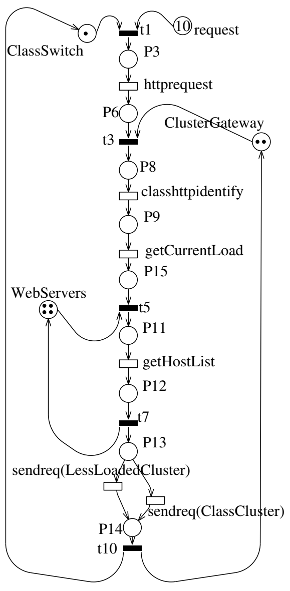


Figure 8. GSPN WSDSAC.

V. Performance Evaluation

The GSPN obtained in section IV should be a mean to evaluate DARC performance properties. But before any analysis, we want to recall the three strategies considered in [1] to evaluate the benefits of the DARC dynamic reconfiguration:

- *DARC-1*: without using the *DynamicThreshold Agent* or any learning mechanism.
- *DARC-2*: *DARC-1* with the addition of a DynamicThreshold agent to improve fairness in the use of resources.
- *DARC-3*: *DARC-2* with the addition of an UpdateManager agent, our full-fledge self-reconfiguration proposal.

Note that these strategies are well-identified in the CSM model in Figure 6, where DARC-3 is indeed represented by the whole CSM, being the other two strategies just subsets of the former. Therefore, the translation of these three CSM yielded the three PNs identified in Figure 7 as well. The results in [1] for these three experiments should prove the GSPN analysis results, analysis that will be carried out using the GSPN exact analysis techniques implemented in [10]. Regarding the GSPN experiments, we used the same number of HTTP requests, i.e., the same load, as in the experimental tests. The rest of the parameters were explained in previous sections and had their counterpart in the experiments in [1].

The results in Figure 9 present the system response time obtained from the GSPNs analysis and they helped to achieve the pursued validation. The results for the three experiments (*DARC-1, DARC-2* and *DARC-3*) are very similar, as it happened in [1]. The response time increases when the reactivity coefficient ρ_{ki} increases, but in compensation the number of requests rejected (not shown in the figure) decreases, which proves the effectiveness of the solution, as it was also shown in [1]. In fact, this is a trade-off in the system, since a change in the reactivity coefficient ρ_{ki} may change the system operating mode (e.g., exclusive or shared mode). The system load (between 50 and 550) was parameterized in the UML model through the *PApopulation* tag and it has its counterpart in the *http request* place of the GSPN. The response time was calculated in the GSPN as the inverse of the throughput given by transition $t17$.

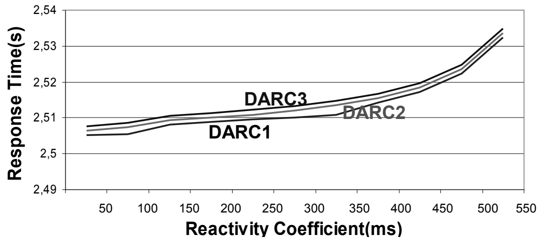


Figure 9. Response time: DARC-[1-3].

From now on, we could consider the different GSPN models as valid and then test the system in hypothetical situations. Figure 10 presents the response time given by the performance models for WS-DSAC and DARC-3 in different situations: (I) presents results obtained from the GSPNs, (II) results obtained in the real environment and (III) evaluates the system using GSPNs when the number of clusters and web servers on the platform WSDSAC is increased. The results obtained using GSPNs differ -1,43% (DARC Architecture) and -1,37% (WSDSAC Platform) from the results obtained in the real environment. We can observe in Figure 10 that the response time for the DARC architecture is a little larger than for the WS-DSAC platform. However, the DARC architecture meets more requests and the rejection rate is lower (as the next figure will show), which proves the effectiveness of the solution.

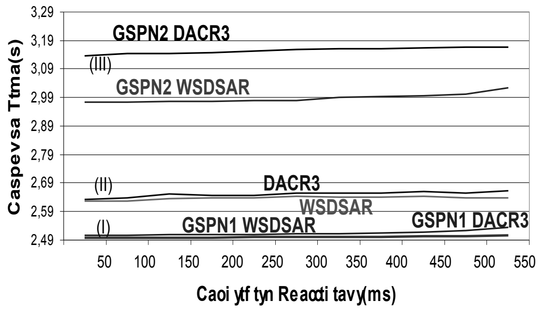


Figure 10. Response time: DARC and WSDSAC.

Figure 11 presents the rejected rate given by the performance models (GSPNs) of DARC and WSDSAC. The rejection rate of the WS-DSAC platform is 6%. However, for the DARC architecture it is only 0.02%. This figure thus shows the advantage of a strategy of load balancing with self-reconfiguration (DARC) over an approach without self-reconfiguration (WSDSAC).

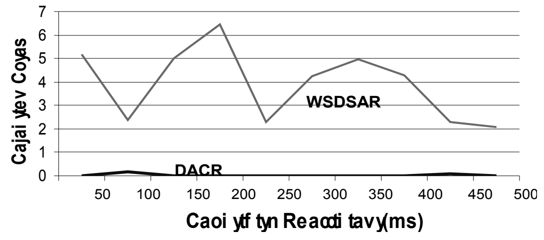


Figure 11. Rejection rate.

VI. Related Work

Due to the focus of this paper, we will restrict the literature review to some works that share with ours the performance study of systems that use mobile agents (MA) or multiagent systems (MAS) as technology, even if they do not concern about web server clusters management.

The work in [11] is likely the closest to ours regarding the techniques used to analyze performance, since it uses SPT, CSM and Petri nets. However, although in the MA scope, it is devoted to analyze and compare the performance of tracking strategies in MA platforms. The work in [12] presents a performance analysis of MA in a distributed network environment, but in this case to compare the performance of SNMP and MA methods to manage a domain of devices.

In the field of MAS, Jurasovic and colleagues [13] evaluate the performance of the well-known Grasshopper and Jade agent platforms. Their analysis focusses on the measurement of average round trip times needed for circular exchanges of messages, the data transferred during a certain period of time, and the message overhead. Nagwani in [14] highlights various parameters associated with the performance of MAS. Then, he studies various models of MAS to discover the impact of such parameters on them. The work suggests an interesting profiling technique to keep track of performance related data in MAS. An important difference of these works with ours is that we not only follow an experimental approach. We use experimental results to validate those obtained with well-known SPE techniques [3], thereby our approach allows modelers to test the system also in hypothetical situations.

VII. Conclusions and Future Work

In this paper, we have studied the performance analysis of the DARC (Dynamic Architecture for Reconfiguration of Web Servers Clusters) architecture [1], an agent-based self-reconfiguration approach for clusters of web servers. Instead of an experimental approach, we have used the SPE techniques [3] to model and analyze this architecture. This has allowed us to validate experimental results previously obtained in [1]. The analytical results obtained are very similar to the ones obtained experimentally (only increasing slightly in one of the experiments), which proves the validity of the models. Moreover, we have also used our models to compare the strategy of load balancing without self-reconfiguration with the approach with self-reconfiguration.

As future work, we plan to analyze other aspects of DARC by evaluating it in a variety of scenarios and conditions. The performance models that we have obtained will allow us to perform an evaluation with much less effort than it would be needed in a real experimental environment.

Acknowledgment

Supported by the project DPI2006-15390 of the Spanish MST and the CICYT project TIN2007-68091-C02-02.

References

[1] C. Marques, S. Ilarri, and G. Barroso, "DARC: A dynamic architecture for reconfiguration of web servers clusters using multiagent systems," in *5th Intl. Conf. on Networking and Services (ICNS'09)*. IEEE, 2009, pp. 169–174.

[2] M. Wooldridge, *An Introduction to MultiAgent Systems*. Wiley, 2002.

[3] C. Smith and L. Williams, *Performance Solutions*. Addison-Wesley, 2001.

[4] *Unified Modeling Language (UML) Resource Page*, OMGs, http://www.uml.org. Last access: 6 December, 2009.

[5] *UML Profile for Schedulabibity, Performance and Time Specification*, OMG, http://www.uml.org, Last access: 6 December 2009, 2005.

[6] D. Petriu and M. Woodside, "An intermediate metamodel with scenarios and resources for generating performance models from UML designs," *Software and Systems Modeling*, vol. 6, no. 2, pp. 163–184, 2007.

[7] A. Serra, K. Cardoso, G. Barroso, and R. Ramos, "Controle de admissao e diferenciacao de servicos em clusters de servidores web," in *Proceedings of the SBRC-SBC*, 2005.

[8] M. A. Marsan, G. Balbo, G. Conte, D. S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*. John Wiley Series in Parallel Computing - Chichester, 1995.

[9] "The CSM to GSPN translator," http://webdiis.unizar.es/~jmerse/csm2pn.html. Last access: 6 December, 2009.

[10] "The GreatSPN tool," http://www.di.unito.it/~greatspn. Last access: 6 December, 2009.

[11] E. Gómez-Martínez, S. Ilarri, and J. Merseguer, "Performance analysis of mobile agent tracking approaches," in *6th Intl. Wk. on Software and Performance*. ACM, 2007, pp. 181–188.

[12] C.-Y. M. J. Holt, A. Huang, "A performance analysis of mobile agents," *IET Communications*, vol. 1, no. 3, pp. 532–538, 2007.

[13] K. Jurasovic, G. Jezic, and M. Kusek, "A performance analysis of multi-agent systems," *Intl. Transactions on Systems Science and Applications*, vol. 1, no. 4, pp. 335–341, 2006.

[14] N. Nagwani, "A performance measurement analysis for multi-agent systems," *Intelligent Agent and Multi-Agent Systems*, pp. 1–4, 2009.