2009 Fifth International Conference on Networking and Services

DARC: A Dynamic Architecture for Reconfiguration of Web Servers Clusters Using Multiagent Systems

Carla Katarina M. Marques
Department of Informatica
University of Estado R.G. do Norte (UERN)
Mossoro-Brazil
carla.katarina@gmail.com

Sergio Ilarri
Department of Inf. Ing. de Sistemas
University of Zaragoza (UNIZAR)
Zaragoza-Spain
silarri@unizar.es

Giovanni C. Barroso
Department of Fisica
Federal University of Ceara (UFC)
Fortaleza-Brazil
gcb@fisica.ufc.br

Abstract

Allocating resources in a web server cluster is usually performed nowadays by a cluster administrator. However, due to the dynamics of the Internet as far as the use of resources is concerned, such task may be considered critical and inefficient if accomplished manually. This paper presents an architecture that performs an automatic dynamic reconfiguration using a multi agent system. The conception, specification and implementation of the architecture are presented. Moreover, several experimental results prove the efficiency and the interest of the proposal.

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.

Within the context of cluster computing, load balancing is an important topic of research. Thus, it is highly desirable that all the tasks executing in a cluster perform as fast as possible. Therefore, situations where some workstations complete their tasks before others and become idle (either due to the fact that the load is not equally distributed or because some computers are faster than others) should be avoided.

Even for experienced cluster administrators, the management of a cluster is an exhausting job. Particularly, allocating the cluster's resources by hand can easily become unmanageable because the processing requirements can change very quickly in a dynamic environment such as the Internet. With this motivation, this paper presents an architecture that performs an automatic dynamic reconfiguration of clusters of web servers using a multiagent system. The main contributions of this work are:

1) We define an architecture to manage web server clusters at the level of planning (capacity management).
2) We propose a multiagent system [1] that self-reconfigures the architecture automatically without the need of a cluster administrator. This system is responsible for keeping the overloading under control without compromising its operation. The agents in the system take dynamic allocation decisions, based on information collected by interacting with the environment, to improve the efficiency of the use of available resources.

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 WSDSAC load balancing architecture, which is the basis of our work. In Section 4, we present our approach for dynamic reconfiguration. In Section 5, we present several experiments that show the interest of our proposal. Finally, we present some conclusions and future perspectives in Section 6.

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 [2], which presents the platform *WSDSAC*, that 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 [3] 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

- $Rexcl$ contains the maximum reactivity coefficient value of the shared work mode. If the ρ_{ki} value is less than or equal to $Rexcl$ value, the cluster works in the *shared mode* and attends requests of different classes. If the ρ_{ki} value reaches the $Rexcl$ value, then the cluster switches to *exclusive mode* and it just accepts requests of the class assigned to that cluster.
- $Rsat$ contains the maximum reactivity coefficient value of the exclusive work mode. If the ρ_{ki} value reaches the $Rsat$ value, then the class switches to *saturated mode* and no longer attends any request.

The two main problems of the WSDSAC architecture 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. On the contrary, the DARC architecture enables the WSDSAC to continuously meet the requests, thanks to the reconfiguration of the resources or threshold updates.

4. The DARC Architecture: Dynamic Reconfiguration

In this section, we describe the *DARC* (*Dynamic Architecture for Reconfiguration of Web servers Clusters*) architecture proposed in this paper. The main objective is to perform a self-reconfiguration of the resources in a web server without the need of a cluster administrator, using a multiagent system. A multiagent system is a natural approach to perform a dynamic management of resources in a distributed way. 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 WSDSAC architecture 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 (e.g., by allocating a host to a cluster that is overloaded) and the classes thresholds ($Rexcl$ and $Rsat$, as explained in Section 3) appropriately. Figure 3 shows the placement of the DARC agents on the different elements in a cluster.

As we observe in Figure 4, the cluster administrator initially defines different classes of services as well as the characteristics of each class, using the *class definition service*. 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.

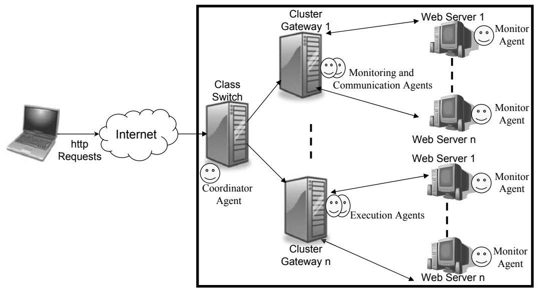

Figure 3. Agents in the DARC architecture.

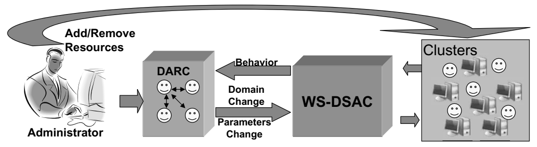

Figure 4. Dynamic capacity management.

Figure 5 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 WSDSAC.

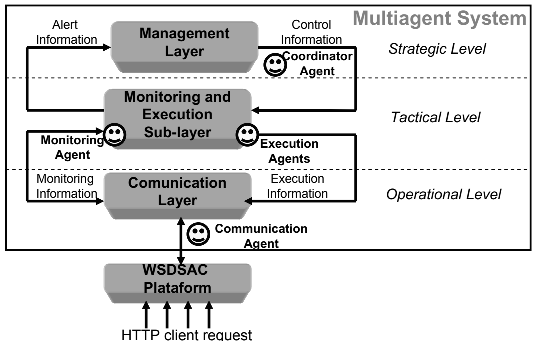

Figure 5. Levels and layers in DARC.

As shown in the previous figure, the multiagent architecture proposed in DARC is composed of different types of agents:

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

must be processed). Due to its importance for the work presented in this paper, we describe WSDSAC in more detail in Section 3.

- Several works use agent technology to perform dynamic load balancing [4], [5], [6], [7]. The main limitation of these works is that they do not perform an automatic cluster reconfiguration (i.e., they do not allocate/deallocate dynamically machines to clusters as needed). 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. Thus, [8] is in this sense a very related work to ours as it also performs a reconfiguration of clusters using agents, aiming at providing a scalable and highly available distributed heterogeneous platform. However, it defines a proprietary operating system called Fire Phoenix; although it can be installed on top of another operating system, having two different kernels introduces an additional overhead. In the work presented in [9], 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. As another example, [10] 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.

Finally, [11] presents a resource allocation strategy to harness idle cluster resources to execute grid applications. The focus of this work is not load balancing or dynamic reconfiguration of clusters, but to exploit available resources in a grid to execute tasks submitted by external users (that may be interrupted if other tasks from local users need those resources). As opposed to the previous works, we propose a multiagent system that performs an automatic dynamic reconfiguration of clusters of web servers.

3. The WSDSAC Architecture: Load Balancing

WSDSAC (*Web Servers - Differentiated Services Admission Control*) [2] performs load balancing by considering different operating modes for the servers in a cluster. Three basic elements can be distinguished (see Figure 1):

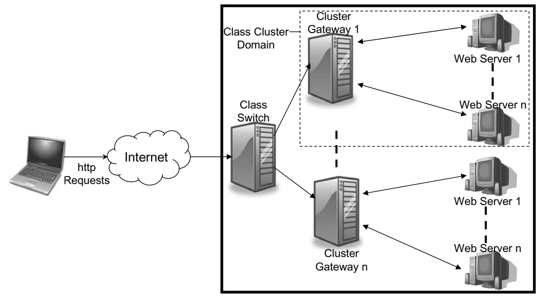

Figure 1. Overview of WSDSAC.

- The *Class Switch*. 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. Requests are grouped in *classes* (defined by the administrator) according to their priority (e.g., in an e-health system HTTP requests from patients that need special care could have higher priority).
- The *Cluster Gateways*. The less loaded Cluster Gateway [1] chooses the less loaded web server to process the request sent by the Class Switch. Each cluster is dedicated to a single class, although it can attend requests from other classes if it is running in shared mode (explained later).
- The *Web Servers*. They process the HTTP requests forwarded by the Cluster Gateways.

The changes in the operating modes of the servers are performed according to a process summarized in Figure 2. Different variables control this architecture:

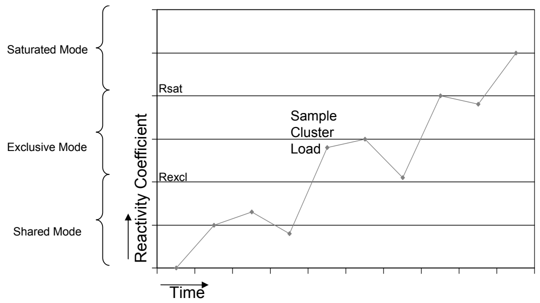

Figure 2. Operating modes in WSDSAC.

- ρ_{ki} is the estimated load of the cluster in the period ki (estimated value of the *reactivity coefficient* during that period).

1. The load of a Cluster Gateway is computed as the average of the loads of its web servers.

- 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 a 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 receives control information from its Coordinator Agent 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) and δ_1 (update parameter) to control the strategy adopted by the DARC architecture. The basic aspects of this mechanism is summarized in Figure 6. 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 (i.e., C2 will allocate own resources in the proportion of δ_1 to process requests for C1).
- 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 $1.5 * Rsat$ 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 %, respectively, and the value of δ_1 to 10 %. These values provided good results in a variety of experiments.

```
If(there exists a cluster in shared mode)
 then do nothing
else if($medLoad \geq \gamma_{low} * Rsat$ and $medLoad < \gamma_{high} * Rsat$)
  then the **MaximumLoad Agent** of the cluster that is overloading
  allocates the less load host of the system to that cluster;
else if ($medLoad \geq \gamma_{high} * Rsat$)
  then the **DynamicThreshold Agent** of the cluster in saturated mode
  increases the thresholds of that cluster by $\delta_1$.
```

Figure 6. Basic mechanism for DARC

5. Experimental Evaluation

To evaluate our proposal, we have performed several experiments in a real environment[2]. As Figure 7 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:

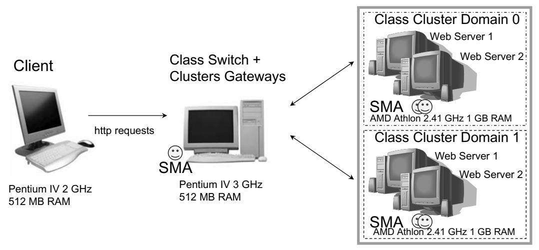

Figure 7. Hardware for the experiments.

- 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 two different clusters/classes. The platform administrator associates each class of service to a maximal load value in the platform through a Class Definition. In these experiments the following values were chosen for each of the two clusters: 300 and 600 for R_{ac} and 210 and 420 for R_{max}

2. Each experiment is repeated several times and average results are reported, checking that the averages are significant.

(i.e., Cluster 1 has more resources allocated because requests of Class 1 have a higher priority). These initial values were determined through an extensive experimental evaluation performed within the context of WSDSAC [2]: They lead to a good performance in a variety of scenarios.

First, we evaluate the WSDSAC architecture, without the DARC architecture, for dynamic reconfiguration. Figure 8 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 $Rsat$ at time instants 120 ms and 270 ms, no request was rejected because Cluster 1 was in shared mode. However, at time instants 180 ms, 430 ms, and 830 ms, Cluster 0 exceeds its threshold $Rsat$ 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 $Rsat$ at the end of the simulation while Cluster 0 is in exclusive mode, leading to rejections of the requests for Cluster 1.

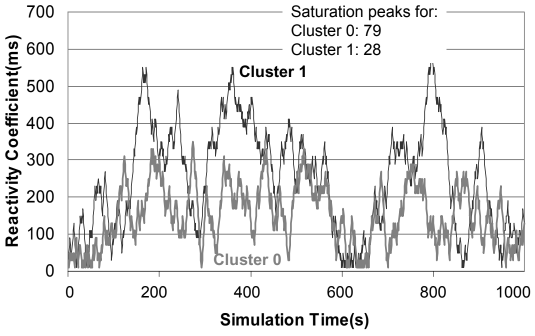

Figure 8. WSDSAC without DARC.

Second, we evaluate the benefits of dynamic reconfiguration with the DARC architecture but without using the DynamicThreshold Agent or any learning mechanism. We will call this approach *DARC-1*. In Figure 9, Cluster 0 exceeds the value of $Rsat$ only at time instant 430 ms; 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 8), the amount of requests rejected decreases. 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 10. At time instants 390 ms and 930 ms, 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). Finally, we perform an experiment with our full-fledge self-reconfiguration proposal (*DARC-3*),

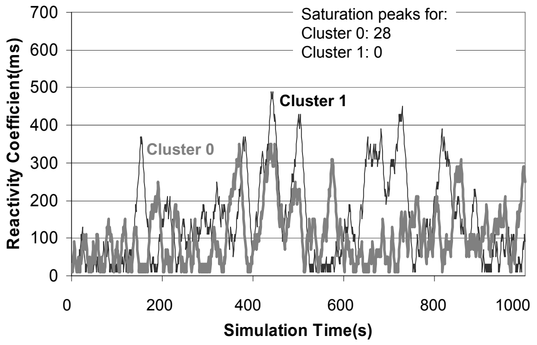

Figure 9. WSDSAC with DARC-1.

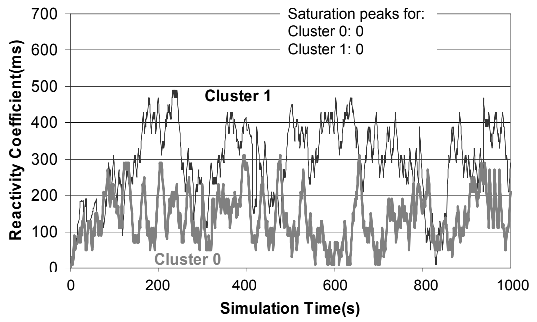

Figure 10. WSDSAC with DARC-2.

where an UpdateManager agent (that learns from the past history) is added. The results are shown in Figure 11. In this experiment, there were no peak saturation for the Clusters 0 and 1. With the utilization of the DARC architecture we can resolve the main problems of the WSDSAC architecture presented to meet the requests. Therefore, we reduce the peak saturation of the system and as consequence the rejected requests. A final comparison between the approaches

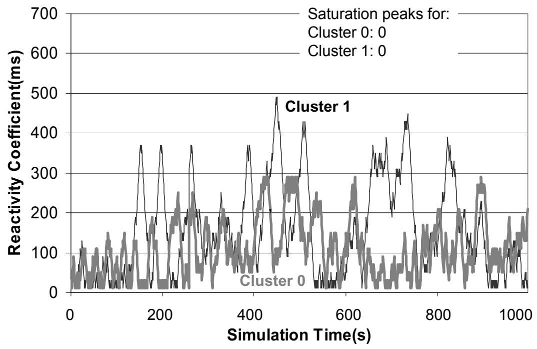

Figure 11. WSDSAC with the whole DARC.

analyzed (*no DARC*, *DARC-1*, *DARC-2*, and *DARC-3*) is presented in Figure 12, where we show the percentage of time when Cluster 0 is under the given load (for different intervals of the reactivity coefficients: 0-100, 100-200, etc). We can observe that with the DARC-3 strategy the load of the clusters is lower most of the time, as this strategy distributes the available resources more efficiently. Thus, with DARC-3 the load is between 0 and 100 during 40% of the time, between 100-200 during 28% of the time, and finally between 200 and 300 during 32% of the time (the load is never above 300). The corresponding figure for Cluster 1 is omitted due to space constraints (a similar behavior is observed).

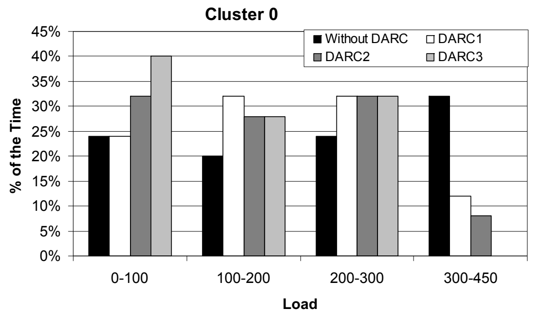

Figure 12. Comparison of Strategies: Load variation in Cluster 0.

6. Conclusions and Future Work

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, no administrator is required to allocate resources or perform complex configuration tasks. The proposed architecture has been implemented in Java and evaluated through several tests that show the interest of the proposal. As future work, we plan on analyzing the possibility to apply this proposal (probably with some extensions) in a different domain (not in the context of web servers).

References

[1] M. Wooldridge, *An Introduction to MultiAgent Systems*. John Willey and Sons Ltd, February 2002.

[2] A. Serra, D. Gati, G. Barroso, R. Ramos, and J. Boudy, "A load-balancing distributed platform based on differentiated services for a telecare application," *International Conference on Control Aplications*, pp. 69–74, September 2004.

[3] R. Olejnik, A. Bouchi, and B. Toursel, "An object observation for a java adaptative distributed application platform," *International Conference on Parallel Computing in Electrical Engineering*, pp. 171–176, September 2002.

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

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

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

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

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

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

[10] C. Adam and R. Stadler, "Adaptable server clusters with QoS objectives," *International Symposium on Integrated Network Management*, pp. 149–162, May 2005.

[11] M. A. S. Netto, R. N. Calheiros, R. K. S. Silva, C. A. R. D. Rose, C. Northfleet, and W. Cirne, "Transparent resource allocation to exploit idle cluster nodes in computational grids," *International Conference on Grid Computing*, pp. 21–30, March 2005.