Applied Soft Computing 48 (2016) 716–728

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A low-level resource allocation in an agent-based Cloud Computing platform

Javier Bajo[b], Fernando De la Prieta[a,*], Juan M. Corchado[a], Sara Rodríguez[a]

[a] Department of Computer Science and Automation Control, University of Salamanca, Plaza de la Merced s/n, 37008 Salamanca, Spain[1]
[b] Department of Artificial Intelligence, Technical University of Madrid Bloque 2, Despacho 2101, Campus Montegancedo, Boadilla del Monte, Madrid 28660, Spain[2]

ARTICLE INFO

Article history:
Received 18 November 2014
Received in revised form 17 April 2016
Accepted 31 May 2016
Available online 5 August 2016

Keywords:
Allocation computation resources
Load balancing
Multiagent systems
Virtual organizations
Linear programming

ABSTRACT

The distribution of computational resources in a Cloud Computing platform is a complex process with several parameters to consider such as the demand for services, available computational resources and service level agreements with end users. Currently, the state-of-the-art presents centralized approaches derived from previous technologies related to cluster of servers. These approaches allocate computational resources by means of the addition/removal of (physical/virtual) computational nodes. However, virtualization technology currently allows for research into new techniques, which makes it possible to allocate at a lower level. In other words, not only is it possible to add/remove nodes, but also to modify the resources of each virtual machine (*low level resource allocation*). Thus, agent theory is a key technology in this field, allowing decentralized resource allocation. This innovative approach has undeniable improvements such us computational load distribution and reduced computation time. The evaluation was carried out through experiments in a real Cloud environment, thus proving the validity of the proposed approach.

1. Introduction

The technology industry is presently making great strides in the development of the paradigm of *Cloud Computing* (CC) [28]. As a result, the number of closed and open source platforms, both of which have similar architectures, has been rapidly increasing [20,30]. From an external point of view, the three most widely known services are *Software* (*SaaS*), *Platform (PaaS)* and *Infrastructure* (IaaS) [32]. From an internal point of view, the services generally offered are considered elastic [16]; in other words, it is possible to have the same level of quality (response time, quality of results, etc.) regardless of the instant of demand. To do so, the amount of resources allocated to these services needs to be variable.

This new model of production and distribution of services is largely possible due to the high number of underlying technologies (virtualization, server farms, web services, web portals, etc.) which have reached their prime [13]. In fact, the hardware infrastructure of a CC platform is one of the most complex and unpredictable technological systems in existence, housing varied equipment such as servers, network modules, uninterrupted power supply units and a long high performance computer infrastructure. Additionally, these components tend to be highly heterogeneous due to the incessant advances in electronics [39], which give way to a new generation approximately every 12–18 months [6]. In order to work, the heterogeneity and intrinsic complexity of this type of computing environments make them difficult to manage and maintain over time [7].

In order to deal with these issues, this type of environment currently uses virtualization technology [52] to abstract the inherent complexity. Virtualization makes it possible to consider the complexity of the underlying hardware abstractly, in the form of virtual machines. A virtual machine is an abstract computational resource that emulates a physical machine with specific software and hardware characteristics. In other words, it makes it possible to create an abstract or virtual, but homogeneous and controllable view of the real hardware [5].

As stated above, these new capabilities derived from virtualization lead to the birth of a new concept: *Elasticity* [16]. This concept is based on the *just-in-time* production method [25], which references the manner in which the (computational) services and the resources they require are produced. Thus, the services produced within the framework of CC only receive the amount of

* Corresponding author at: Department of Computer Science and Automation Control, Plaza de la Merced s/n, 37008 Salamanca, Spain.
E-mail address: fer@usal.es (F. De la Prieta).
[1] {fer, corchado, srg}@usal.es.
[2] {jbajo}@fi.upm.es.

http://dx.doi.org/10.1016/j.asoc.2016.05.056
1568-4946/

resources they need to maintain the level of quality, which is previously agreed to by contract with end users [49,46]. As a result, the main objective within this context is to maintain the level of Quality of Services (QoS) [2], regardless of demand. This objective implies a need to dynamically manage the computational resources assigned to each service offered to the user. In order to do so, these platforms predominantly use a management system (monitor and control) for the underlying computational infrastructure [28].

In this sense, virtualization technology has recently begun to advance at great speeds, allowing the physical characteristics of the virtual machines (memory, CPU, storage space, etc.) to be changed dynamically, even when the machine is running [15]. Nevertheless, existing models have not yet incorporated this new capability offered by the technology [29,38]. These models use a classic approach in which elasticity is based on modifying the number of nodes that attend to a particular service. The hypothesis of the present research work is based on the premise that since technology is able to offer new capabilities, it is necessary to design new resource allocation models that take these new characteristics of the technology into account. In this context, the *Theory of Agents* and *Multiagent Systems* (MAS) [50] can provide a new route for managing CC systems based on the distribution of responsibilities, flexibility and autonomy. Managing the functions of the nucleus of a CC system through an agent-based model allows the resulting platforms to be much more efficient, scalable and adaptable than they currently are [43,41].

This work proposes a low-level resource distribution model in which a level of resources is associated with each individual virtual machine. This calculation model is based on agent technology and, as such, is a distributed model, thus making it possible to distribute computational resources throughout the entire CC environment and allow the distribution of its complexity and associated computational costs. This new approach in monitoring and, in particular, controlling the CC system, makes it possible to incorporate the new characteristics, as previously mentioned, that virtualization has to offer.

This document is organized as follows: the following section provides a detailed description of the problem, Section 3 presents the current literature related to the problem. Then, Section 4 proposes a solution based on multiagent systems, while the evaluation and validation of these systems are presented in Section 5. Finally, the last section presents the conclusions of the research.

2. Problem statement

2.1. Cloud environment overview

Given the complexity of the environment, as well as the different artificial and human components involved in this context, it is necessary to define how the services will be offered at a technical level. To do so, a CC is presented in Fig. 1, in which each software service for the platform, at the PaaS or SaaS level, can be deployed simultaneously on various virtual machines (computational nodes or *workers*).

Each Physical Resource or server (PR) will usually host a set of Virtual Machines (VM). This will allow every physical server in the system to have an associated matrix at all times with information regarding the state of the real hardware (namely, the physical machine) as well as the different virtual machines that it hosts at any given time. This matrix, shown in Eq. (1), is a representation of the state of each physical server that has been launched and can be used to analyze the allocation of resources to each service and the amount of resources to be lent.

$$e_i = \left\{ \begin{array}{c} \mathrm{PR^e} \\ \mathrm{VM_1} \quad \ldots \quad \mathrm{VR_m} \end{array} \right\} where$$
$$PR^e = \left\{ hostname, IP, mac, state, M_{\max}, vcpu_{max}, M_{min}, vcpu_{min}, benchmark \right\}$$
$$VM_i = \left\{ IP, state, M, vcpu, M_i, \overline{p_{cpu}} \right\} \quad (1)$$

The intrinsic characteristics of the service are therefore determined by the template, which is used to instantiate a virtual machine that offers a concrete service and contains the software and the minimum hardware characteristics of each node associated to the concrete service. Once the virtual node has been instantiated or created using the template, it is possible to modify, in execution time, the resources associated with a specific service. Consequently, every template for a virtualized node associated to any service k (VM_t^k), will be described through a set of properties as seen in Eq. (2): identifier, minimum assignable memory, minimum number of assignable CPUs, type (hardware or software service) and state, which determine whether it is balanceable:

$$VM_t^k = \left\{ ID^k, M_{\min}, vcpu_{min}, type, state \right\} \quad (2)$$

This deployment model of the hardware infrastructure, which is based on two levels of abstraction (real and virtual), makes it possible to provide any type of computer services. While the user will consume these services, it will be necessary to have previously reached an agreement about the QoS, through a Service Level Agreement (SLA). Formally, as shown in Eq. (3), the SLA is formalized for any given user j ($ServA^j$) as the combined set of user agreements established for each service i on an individual basis (SLA_i^j) [35,19].

$$\mathrm{ServA^j} = \cup_{i=o}^{i=m} \mathrm{SLA_i^j} \quad (3)$$

Using this simple expression, the goal for achieving an adequate model of the context consists of measuring the quality of services offered. Various related studies can be found in the current state of the art [2,21,15]. Within the scope of this work, the metrics that are directly dependent on the underlying computational resources will be used by the service, and the response time for each request will be selected. In other words, for a service k with a set of methods (r_i^k) which make up the service API, the quality of each request that forms the service API is determined by Eq. (4) for the size of the response (s_i^k) and the corresponding transmission time (t_i^k):

$$\mathrm{QoS^k} = \left\{ \overline{\mathrm{r_1^k}} \cdots \overline{\mathrm{r_i^k}} \cdots \overline{\mathrm{r_n^k}} \right\} \text{ donde } \overline{\mathrm{r_i^k}} = \frac{1}{\mathrm{m}} \sum_{i=1}^{m} \mathrm{s_i^k / t_i^k} \quad (4)$$

Using this very characteristic deployment model, the control system in a CC environment should vary the computational resources assigned to each service according to the demand that exists at any given time, making it possible to maintain the QoS levels in each of those services. In this regard, the greatest advantage of virtualization is that the assignment of resources at any instance of execution can be reconfigured dynamically, which makes it possible to elastically modify the amount of resources associated with each service in execution time. In terms of requests for a specific service, the demand is balanced among the different virtual machines that are associated to the service.

The distribution of resources can be presented from different points of view according to the capabilities permitted by the underlying technology. First, *Distribution at the service level* requires a simple balance of the workload among the worker nodes that are offering a specific service. This balancing has been highly extended from the birth of cluster computational systems in HPC (*High performance Computing*) environments [1,10,44]. Second, *Distribution at the infrastructure level* is a more complex and novel method which

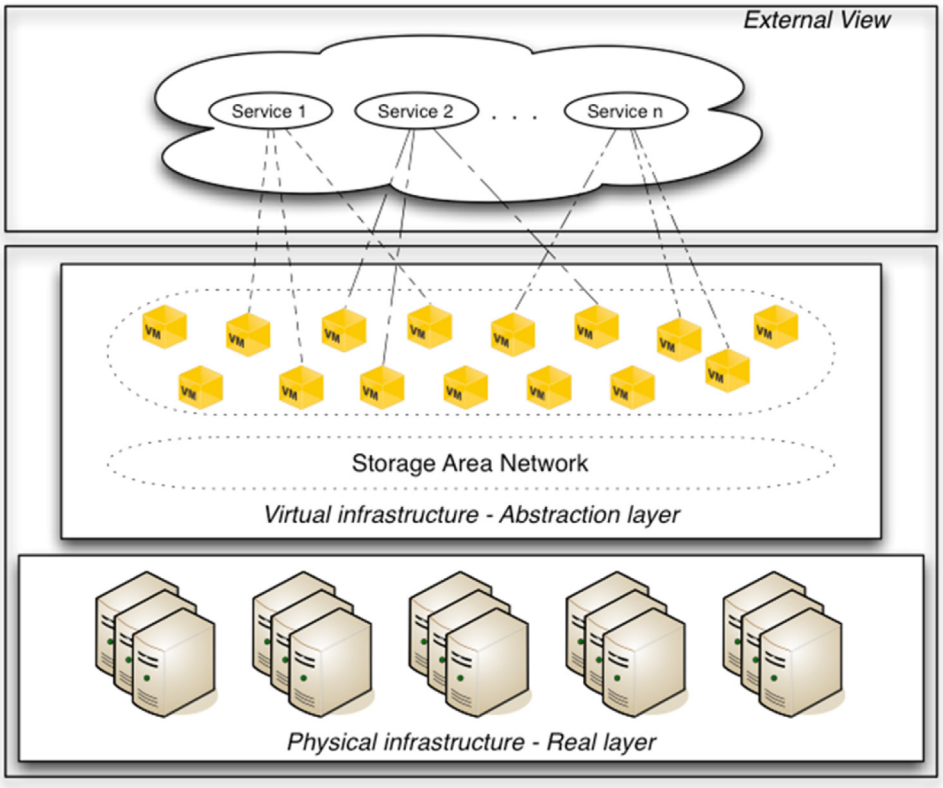

Fig. 1. Deployment model in Cloud Computing.

was made possible after the birth of virtualization technology [52]. This technology permits dynamic and automated tasks by updating the assigned resources and nodes that are offering the services.

However, these methods do not address the fragmentation of resources [3,45]. Normally, a physical server hosts multiple virtual machines, some of them with a saturation of resources and other machines with underutilized resources. In the latter case, the underutilized resources cannot be released because they have been assigned (fragmentation). Previously, during a redistribution at the infrastructure level, the redistribution associated with cluster deployments involved only the modification of the number of nodes that attend to a specific service [8].

Recently, virtualization has also allowed for new possibilities [26], such as dynamically modifying the number of resources associated with each of the execution nodes. As a result, the redistribution of resources can now be seen from two points of view: the micro and macro level. In the (i) micro perspective, the system asks the different physical machines containing the nodes for a specific service to provide more computational resources to each of the nodes, which increases/decreases its performance. The (ii) macro level allows instantiating a new node that can be hosted by an active physical machine in the CC environment, or by one of the machines in sleep mode.

2.2. Problem statement

This paper aims to study new strategies for the allocation of resources at the micro level, where capabilities offered by the underlying technology (virtualization) have recently allowed for the development of design strategies that, unlike current strategies, allocate resources at fine grain (CPU and memory) [4,29]. For example, as presented in Fig. 2, when there are underutilized resources in any virtual machine of a real server, these resources can be internally moved to other virtual machines at runtime, without having to commit new resources globally (such as the instantiation of new nodes, or the migration of the existing virtual machine itself). This capability makes it possible to manage the elasticity at a micro-level inside each physical server. This characteristic extends beyond current proposals that seek to optimize the use of energy in a Cloud environment [40,4] seeking greater efficiency not only at the level of the physical server, but also for each of the virtual nodes, thus minimizing the number of underutilized resources that have been blocked at each virtual node.

As an initial approach, the present work must first create a control and monitoring model in a Cloud environment, which will allow for a more precise and dynamic control of resources. Within this framework, a new model is proposed based on the use of an intelligent organization of agents, an approach that has not been used to a great extent within this field [43]. The use of such a design also requires the design of new distributed algorithms to optimize resource, which is in itself another novel approach with regard to the centralized techniques based on mathematical models [23,36,42].

3. Related works

The assignation of computational resources in CC environments is commonly known as Resource Allocation (RA), which has as its main goal to maximize the benefits of the supplier and meet the needs of the users [45]. Obtaining an optimized solution for RA is a highly complex task which increases exponentially depending on the number of existing resources and services offered. The problem is further aggravated when considering that RA is usually applied on distributed and complex systems such as a large server clus-

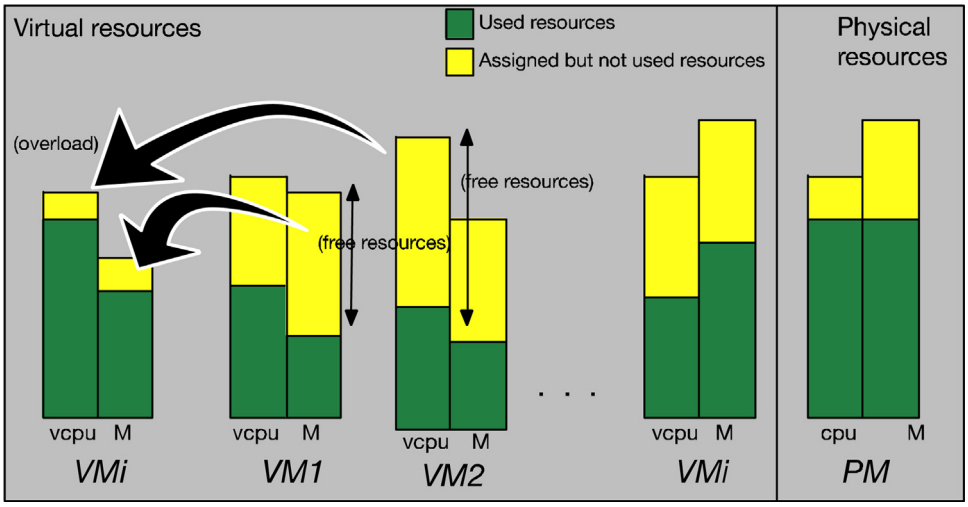

Fig. 2. Example of micro-resource distribution.

ter, data centers or environments Grid. The technology makes it possible to allocate resources at a local level, among each individual service or physical node, or at a global level, among different physical nodes. Consequently, the correct distribution of computational resources in a CC environment is a key decision given that the distribution can maximize the efficiency of the environment and reduce associated costs. This process is commonly referred to as *Resource Allocation Strategy* (RAS), which should be able to minimize any of the following problems [45]: (i) *overused* or *underused* resources; (ii) *fragmentation* of resources,(iii) *r*esource *scarcity*, and (iv) resource *contention.* Therefore, to find a solution as optimal as possible it is necessary to follow a strategy with this purpose.

The current state of the art for RAS provides different approaches which normally involve three concepts [12,45]: (i) the SLA agreements established with the user, (ii) the energy consumption in the CC environment, and (iii) the state of the CC environment itself. Due to its complexity and current prevalence, this problem is of great interest to the scientific community [45,33,29].

Traditionally, RAS has been approached from the point of view of the interests of the market, which give priority to the agreed-upon SLA with users (SLA-based [51]), instead of other variables. However, there are now approaches that prefer to give more importance to energy consumption, although in certain specific situations the system may violate the reached agreements (Energy-based [40,9,4,41]). Regardless of the strategy, the problem in the prior state of the art was dealt with by using different techniques such as [45]: (i) the use of policies for allocating resources, (ii) the ability to migrate virtual machines between physical machines, fulfilling a utility function, (iii) the compliance with a utility function from the environment as a whole, (iv) the use of auctions for resource reservation based on estimated demand, and finally, (v) planning and (vi) resource prediction.

Notable among the techniques that make use of virtualization as a basis for RA is the work by You et al. that proposed the RAS-M framework [53]. It is a model based on a market economy and seeks a redistribution of resources that could give way to a fair market price. Other models use base mathematics such as game theory [48,23,36,47] for the distribution of resources. There are also other authors such as Caton et al. [14] who use social media information to determine the use of resources, and studies that consider energy efficiency by applying energy saving policies in physical or virtual machines [37], or by efficiently distributing virtual machines according to the use of each physical piece of equipment [8]. In addition to these approaches, dynamic and adaptive methods have recently appeared. For instance Malhotra *et* Malhotra [31] propose an adaptive mechanism based on parallel processing, which is based on MAS. Another notable work is proposed by Sriram *et* Ravimaran [42], which combines the planning of virtual resources with forecasting the use of resources.

However, the techniques that have been studied, which are related to virtualization technology, are only based on instantiating, stopping or migrating virtual nodes; they do not exploit one of the latest possibilities offered by virtualization technology: the dynamic modification of virtual machine resources in execution time [15]. The use of these capabilities can be applied in combination with other existing strategies at the macro level, which would reduce the need to address the underutilization and fragmentation of resources in CC environments. In order to address this weakness, this paper proposes an RAS focus on each physical machine and a set of virtual machines that use MAS to allow for the allocation of resources at a micro level.

4. Redistribution resources proposal

As previously explained, the development of a monitoring and management system for a CC environment that follows a MAS-based design model differs from traditional models that control this type of platform. This approach allows the decision-making process to be carried out right where the information is gathered, on a base that provides local knowledge, which has made it possible to design agile control processes based on uncertain information, prior knowledge, and interaction among similar agents. The next subsection will present a brief overview of the multiagent architecture model that has been used to monitor and control a CC system. The distribution of resources at the (micro) infrastructure level will then be presented in detail.

4.1. Multiagent architecture overview

The proposed MAS architecture is based on organizational aspects and, as such, it is necessary to identify the organizational structure to be used. To do so, the first step involves identifying the main components, which permits establishing the interaction model based on an analysis of the needs of the potential system users. Based on this analysis, it was possible to deduce the roles of the users and components that participate in the system and the way they will exchange information. Only a brief explanation of the primary tasks of each of the agents will now be provided, as a

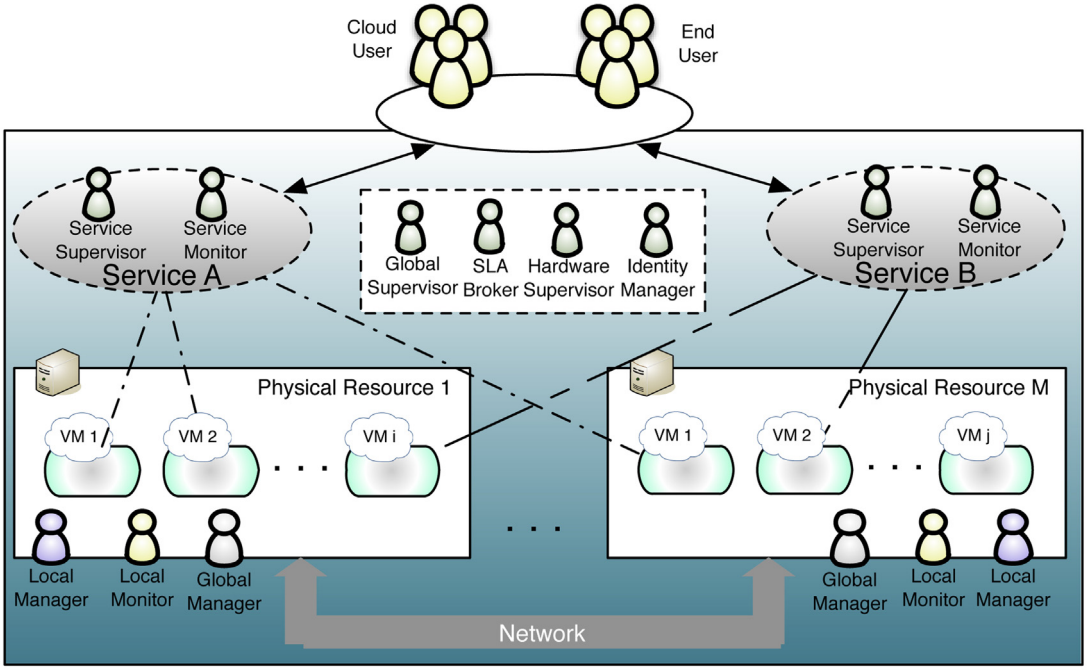

Fig. 3. Agents distributed over the infrastructure.

detailed presentation is not one of the objectives of this research. However, the reader may find more details about the architecture in Refs. [17,18,24].

Each one of the agents/roles that participate in the organization is located throughout the entire computational environment. As shown in Fig. 3, each physical server in the CC environment contains an agent in charge of monitoring (*Local Monitor*) and another responsible for the local level (*Local Manager*). Between the two they have the authority to completely control the Physical Server (PR) where they are located, which in turn implies a distribution of resources in the virtual machine. However, when the resources must be distributed, which involves the assignment or removal of nodes for a particular service, another specialized agent (*Global Manager*), which is also located in each one of the physical nodes of the infrastructure, is in charge of making these types of decision, which involves more than one physical node on the CC platform.

Following a similar model, each service offered to the cloud users is associated with two agents, one for monitoring (*Service Monitor*) and the other for control (*Service Supervisor*), both of which are in charge of ensuring compliance with the previously established SLA agreement. They are physically located in the node that balances the workload among the different worker nodes, which permits them to have precise information available to make the correct decisions at their level. In this sense, the tasks for this level are related to the workload balance among the different nodes, error detection and, most importantly, monitoring the quality parameters for the service.

There are also other tasks that can be done to ensure the proper operation of the CC system. These task are performed by other agents located at the entry point of the CC system. To begin, there are two control agents, the first of which is in charge of controlling the hardware infrastructure (*Hardware Supervisor*), its state, and the starting or stopping of the physical servers according to demand. A supervisor agent is the global controller (*Global Supervisor*), which ensures that the remaining components and agents function correctly and in accordance with their specification. Finally, there is also an agent in charge of establishing service agreements with the platform users (*SLA Broker*), which can negotiate the QoS level of services according to user needs and the state of the system at any given moment.

Using an architecture such as that proposed in this work, it is possible to completely govern a CC system (resource distribution, orchestration of services, agreement negotiation, etc.). Using the architecture as a basis, the following section will present possible models under consideration for the distribution of resources in a CC environment.

4.2. Service and micro level computational resources allocation

The simplest approach to vary the resources assigned to a service is to update the assigned weight to each node that is offering the services. In other words, to balance the weights among working nodes of the service. Although, this approach is the simplest, it has a key role in the new distribution model since it is in charge of ensuring that the SLA agreements are in compliance with regard to the software services agreed upon with the users of the CC platform.

There are two agents in this redistribution specialized in monitoring and supervision tasks (see Fig. 4): the *Service Monitor* and the *Service Supervisor*, as follows:

- The *Service Monitor* agent is in charge of controlling the state of the balancing system. It implements algorithms that calculate the QoS on a continuous basis for each one of the nodes that is being balanced at any moment. To do so, the agent periodically recalculates the weight of each node within the service, applying the following algorithm:
  - Maintain a set of routing groups, each of which is linked with different worker nodes with similar QoS levels.
- The nearest neighbor grouping algorithms is used to determine the QoS groups, based on the calculation of the distance between nodes using the Euclidean distance between quality of service vectors in each node:

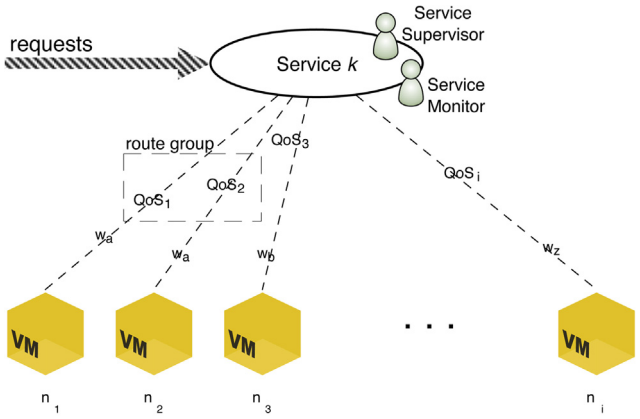

Fig. 4. Redistribution of resources at the service level.

$$QoS_{n_i}^{k} = \left\{ \overline{r}_1^k \cdots \overline{r}_i^k \cdots \overline{r}_n^k \right\} \quad (5)$$

- In order to assign a weight to each group, which is an inexact process, an iterative process is used. Initially, all groups have the same weight; however, it is adjusted progressively according to the demand that each group can accept. Thus, the groups with the best QoS have a higher weight, while those with a lower QoS have a lower weight. After a period of stabilization, the process is repeated starting with the previous point.
- The *Service Supervisor* agent has a key role given that it must determine, according to the data generated by the *Service Monitor*, whether it is necessary to assign more or fewer resources to the service:
  - It must first decide when to remove the subset of resources associated with the service. After removing any execution resources, the remaining resources associated with the service will have to respond to the demands associated with those that have been eliminated. This increase in the workload of each of the nodes has a corresponding period of stabilization in which the *Service Monitor* recalculates the new QoS parameters following the presented algorithm.
- It must first decide when to remove the subset of resources associated with the service. After removing any execution resources, the remaining resources associated with the service will have to respond to the demands associated with those that have been eliminated. This increase in the workload of each of the nodes has a corresponding period of stabilization in which the *Service Monitor* recalculates the new QoS parameters following the presented algorithm.
- More resources are associated with a service when the existing nodes, and the resources that have been assigned to them, are unable to respond to the demand for requests according to the SLA agreements established with the users. In order to do so, a distribution of resources is first initiated at a micro level. If the problem is not resolved at this step, the distribution of resources can be initiated at the macro level. In the current state of the art, there are different studies, as shown in Section 3, which deal with this problem.

4.3. Redistribution resources at the infrastructure (Micro) level

As, previously explained, redistribution from the point of view of the infrastructure at a micro level is managed innovatively by two specialized agents, as shown in Fig. 5, located in each server of the infrastructure. The new process works as follows: firstly, the *Local Monitor* agent is in charge of retrieving information about the state of each virtual machine and the related service. This monitor agent will have its own knowledge of a matrix (Eq. (2)) with the information it needs about the physical server and each of the virtual machines it hosts. Secondly, the *Local Manager* agent can oversee the assignment of resources to manage the resources within the machine by updating the resources (CPU or memory) assigned to each of the virtual machines. With these three functionalities it is possible to redistribute resources within a physical server. The resources that can be redistributed are the number of virtual processors (*vCPU*) and the physical memory (*M*).

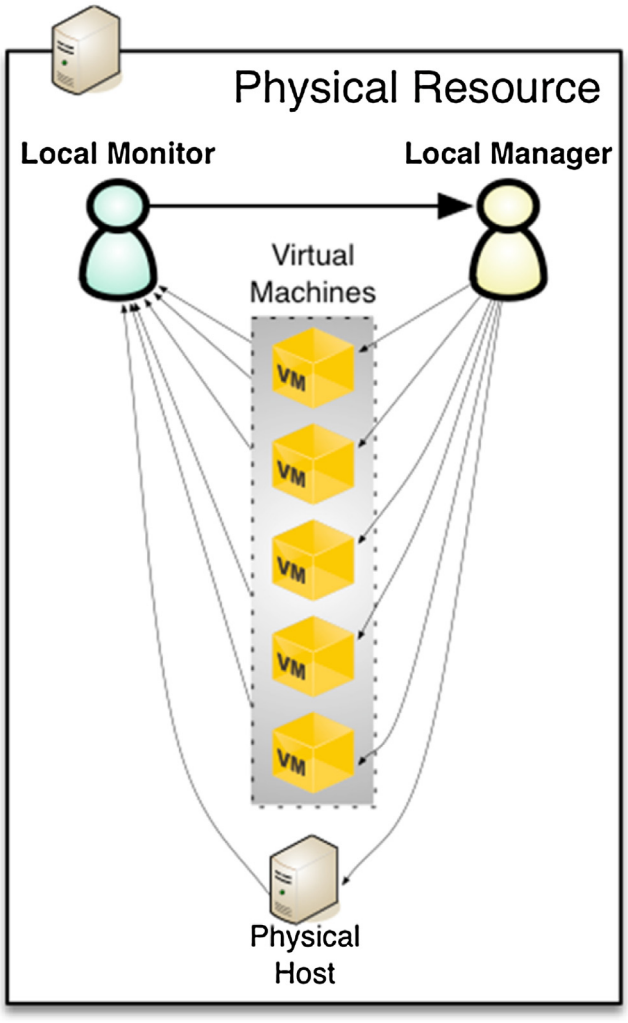

Fig. 5. Agents in charge of resource allocation at micro-level.

The process of distributing resources begins when the *Service Supervisor* agent of the concrete service detects that a particular service has insufficient resources to meet current demand and, as a result, does not comply with the QoS parameters previously established. During this process, it is necessary to take into account the type of service to which the virtual machines, those in execution, belong.

The *Service Supervisor* agent formally initiates the process when it sends a message indicating the need for resources. This message is received by each of the *Local Manager* agents from the physical machine hosting the nodes for the particular service. Each of these machines initiates a parallel process for the reassignment of resources at the micro-level, which results in a global increment in the number of resources assigned to each service.

When the process of distribution of resources is initiated, the *Local Manager* agent for each machine configures a structure of information, as shown in Table 1, which contains all of the data

Table 1
Summary of the information of physical server.

	processing			Memory				General		
	$vCPU_{max}$	$vCPU_{min}$	$vCPU_{used}$	M_{max}	M_{min}	M_{used}	$M_{asigned}$	ID_s	type	priority
VM_1										
$\boldsymbol{VM_2}$										
VM_m										
$\boldsymbol{PR^{e_1}}$										

regarding the assignment of resources in the physical machine. The structure gathers the instantaneous information ($I^t_{e_i}$) for the complete state of the assignment of resources at a specific time t, the current degree of usage for each resource, and the service and priority of the node in the system with regard to balancing the service, as calculated by the corresponding *Service Monitor*. The importance of these three parameters is due to:

- *IDs*: associated to the service identifier, which can determine whether a virtual machine corresponds to an infrastructure or software service node.
- Type: determines whether the services are *stateless* or *stateful*, where *stateless* applications are defined as having various simultaneous instances at execution since they do not store the state of the service. On the other hand, *stateful* virtual machines have only one instance at execution. Therefore, if this service (node) requires more resources, it takes priority over the others, because it is not possible to add more nodes in order to increase the quality.
- Priority: determines the priority of the virtual machine in the process of balancing the work at the service level. Its relevance lies in its ability to measure the performance of a virtual machine with regard to the other nodes for the same service. The measurement corresponds to a value within the range 0 (low priority) to 1 (high priority) and is calculated with the expression $Priority_i = {}^{w_i}\!/\!_{w_{max}}$.

Before describing the process of redistribution of resources performed by the *Local Manager* agent, it should be highlighted, as stated below, that the process only affects the machines that work with software services. Machines with static characteristics are not taken into account during the reallocation process; therefore, of the n machines hosted by a physical server, only m machines, those that deal with software services with dynamic characteristics, are included in the algorithms presented below.

The distribution of resources is different for assigning vCPUs and memory. The following section will first present the distribution process for the *vCPU*, followed by the procedure for reassigning memory.

4.3.1. vCPU allocation algorithm

The goal of assigning *vCPUs* is to not underuse execution resources. If there were any, they could only be assigned to a physical machine. A machine state is shown in Fig. 6. The process of assigning *vCPUs* is initiated by verifying whether the physical machine has *vCPUs* that have not yet been assigned (See Eq. (6)).

$$\mathrm{PR^{e_1}}(\mathrm{vCPU_{max}}) \neq \sum_1^m \mathrm{VM_i}\left(\mathrm{vCPU_{used}}\right) + \mathrm{PR^{e_1}}(\mathrm{vCPU_{used}}) \tag{6}$$

If there are no unassigned *vCPUs*, the agent *Local Manager* iterates over every virtual machine to determine whether the amount of processing that has been assigned is being underused. If the value $vCPU_{used}$ is less than the constant supervised by the administration, one of the cores from that machine is reassigned to another machine that needs more resources.

4.3.2. Memory allocation algorithm

The process of reassigning memory is more complex and, as such, requires a linear programming model to determine the most optimal assignment of resources. The first step is similar to the process of assigning virtual processing units. If the physical server has free (underused) memory ($PR^{e_i}(M_u)$) that can be assigned, the memory is directly assigned to the requesting virtual machine, thus complying with the maximum memory restriction (Eq. (7)) that can be assigned to the template of the virtual machine that corresponds with the service under QoS deficiency.

$$\mathrm{VM_i}\left(\mathrm{M_{asigned}}\right) \leq \mathrm{VM_i}\left(\mathrm{M_{max}}\right) \tag{7}$$

This process of direct assignment is carried out according to the following expressions where $VM_i(M'_{asigned})$ corresponds to the memory previously available in the virtual machine, and is based on the priority that the virtual machine has in balancing the load. The process was modeled this way so as to (i) not greatly modify the distribution of priorities in the previously assigned balance process, which implies more distribution, even at the macro-level, and (ii) not assign all the available resources to only one single virtual instance.

$$\begin{cases} \mathrm{VM_i}\left(\mathrm{M_{asigned}}\right) = \mathrm{VM_i}(\mathrm{M'_{asigned}}) + \mathrm{VM_i}(\mathrm{M_{max}}) * \mathrm{priority_i} \\ \qquad \text{if}\,(\mathrm{VM_i}(\mathrm{M_{max}}) * \mathrm{priority_i}) < \mathrm{PR^{e_i}}(\mathrm{M_u}) \\ \mathrm{VM_i}\left(\mathrm{M_{asigned}}\right) = \mathrm{VM_i}(\mathrm{M'_{asigned}}) + \mathrm{PR^{e_i}}(\mathrm{M_i}) \\ \qquad \text{if}\,(\mathrm{VM_i}(\mathrm{M_{max}}) * \mathrm{priority_i}) \geq \mathrm{PR^{e_i}}(\mathrm{M_u}) \end{cases} \tag{8}$$

If there is no free memory on the physical server, it must be reassigned from other virtual machines; in other words, underused memory (M_u) from a set of instances of execution must be taken and reassigned to the requesting machine. Thus, at a technical level, it is possible to redistribute the memory that is being underused while maintaining a margin of security (M_l).

Thus, the underutilized memory of each virtual machine is given by Eq. (9):

$$\mathrm{VM}(\mathrm{M_u}) = \mathrm{VM}\left(\mathrm{M_{asigned}}\right) - \mathrm{VM}(\mathrm{M_{used}}) \tag{9}$$

And therefore, all underutilized memory from all physical servers is given by the following expression (Eq. (10)):

$$\mathrm{PR^{e_i}}(\mathrm{M_u}) = \sum_1^m \mathrm{VM_i}(\mathrm{M_{assigned}}) - \mathrm{VM_i}(\mathrm{M_{used}}) \tag{10}$$

The problem lies in how to redistribute the underused memory among the different virtual machines so that at the end of the process, the amount of underused memory of the server $PR^{e_i}(M_u)$ is reduced. This is an optimization problem of the underused memory; the solution is obtained by using linear programming so that there are m independent variables that correspond to the new memory assignment. The function that must be minimized is composed of the sum of the underused memory that will be assigned to each virtual machine according to the actual percentage of underutilization; this makes it possible to determine what can be gained by assigning memory in each case (Eq. (9)).

$$\mathrm{MIN}PR^{e_i}(\mathrm{M_u}) = \sum_1^m \mathrm{x_i}\left({}^{\mathrm{VM_i}(\mathrm{M_u})}\!/\!_{\mathrm{PR^{e_i}}(\mathrm{M_u})}\right) \tag{11}$$

Where x_i is the amount of reassigned memory of each virtual machine. This minimization model has the following two restrictions:

- The sum of the objective variables will be equal to the sum of the initial underused memory (Eq. (10)).

$$\sum_1^m \mathrm{x_i} \leq \mathrm{PR'^{e_1}}(\mathrm{M_i}) \tag{12}$$

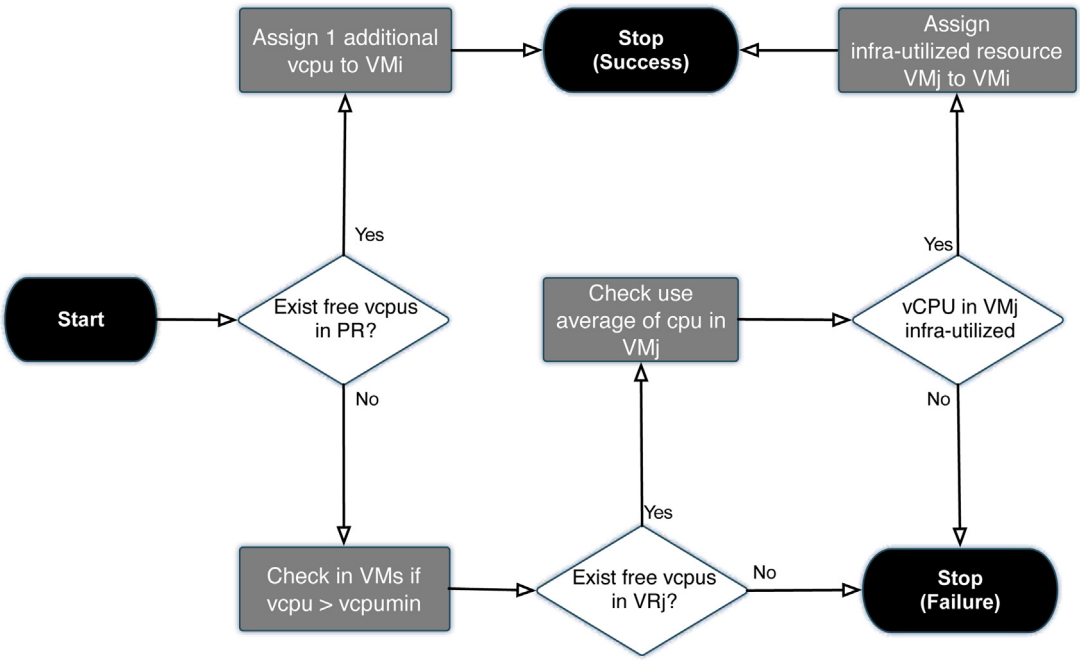

Fig. 6. Reallocation of resources (*vCPU*) at micro level.

- It will also be necessary for each variable x_i to be greater than the margin of security.

$$x_1, \ldots, x_m > M_l \tag{13}$$

Once the problem is solved, it is possible to obtain the value of memory allocated to each virtual machine, as given by following expression:

$$VM_i\left(M_{asigned}\right) = VM_i(M'_{used}) + x_i \tag{14}$$

Once the new distribution of memory has been determined, if the results obtained during the process of calculating the new assignment are greater than the resources of the service machine that initiated the change, then the new assignment will be applied by the *Local Manager* agent, and the *Service Supervisor* agent associated to the service that initiated the process will be informed that the node has received a new resource assignment. If local distribution is not possible, the Service Supervisor is notified. This agent must evaluate the different results obtained from each physical node where the service is deployed and, if necessary, it initiates the process of resource allocation at a larger level (macro distribution).

4.3.3. Micro level allocation resources considerations

Finally, it should be noted that this process can be initiated by various services simultaneously; in this case, the *Local Manager* should take the following considerations into account:

- The process of calculating new resources does not depend on the type of service but on the priority of the machines within each service, and on the resources that the service is using at any given time.
- Thus, if two or more services request an increase in resources at the local level, the proposed changes will only be applied if:
- The machine has available resources that can be assigned to the resources.
- The machine has no available resources; a change of resources will only occur if, for all the services that have requested an increase of resources, the calculation of the new assignment increases the resources that have been assigned to each node associated with the services that have requested more resources.
- The machine has no available resources and no adequate solution was found in step two for the entities requesting resources. In this case, the agent initiating the process, the *Service Supervisor*, will be informed that there are insufficient resources; it must therefore take any measures deemed necessary.

In any case, once a change in resources has been applied, it will not be possible to readjust the services again until the quality of services has been stabilized and updated after the distribution. The wait time between two reassignments is a constant that will be supervised by the administrator in the CC environment.

5. Evaluation: case study

The evaluation of the proposed dynamic distribution model is a thorough task and requires a hardware and software environment specially adapted to its needs. The evaluation and validation of the model for this research work will be done through a CC platform developed within the scope of the research done by the BISITE research group,[1] and will include different computational services at the hardware and software level. From the beginning, this platform was conceived to integrate the proposed MAS. In the case study used, this CC platform was deployed in the HPC environment of the BISITE research group and composed of 15 latest generation machines (Intel core i5 processors and 16Gb of memory RAM) that support virtualization in the hardware with the use of Intel-VT technology and the KVM (Kernel-based Virtual Machine) virtualization system.

Regarding the implementation of the MAS, the different components of the system were implemented using the Python language (Version 2.7). The module PSutil[2] was used to generate statistics. Communication among agents is done by using AMQP protocol[3]

[1] http://bisite.usal.es/en.
[2] https://github.com/giampaolo/psutil.
[3] http://www.amqp.org/.

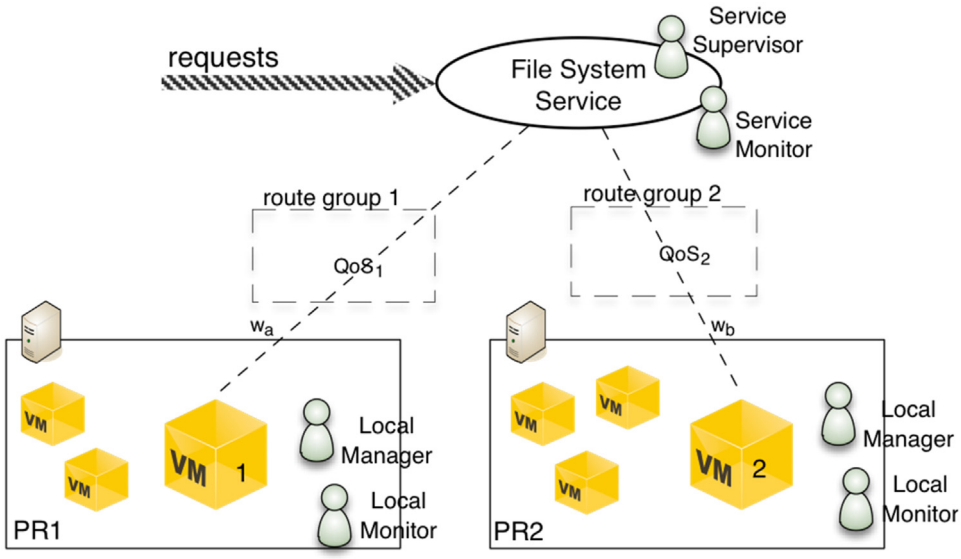

Fig. 7. Initial state of the evaluation case study.

(*Advanced Message Queuing Protocol*), sending messages in JSON format (JavaScript Object Notation), using the server message RabbitMQ.[4] The operating system is based on debían. Finally, for interaction with KVM both the API Libvirt[5] and KVM commands are used when Libvirt does not work properly.

In order to evaluate the proposed multiagent architecture, a series of experiments were conducted with the aim of simulating the behavior of an organization and its members in a real adaptation case. The results obtained from these experiments have made it possible to empirically evaluate whether the dynamic system responds according to its specification, dynamically adapting according to the state of the environment and the demand for services. Upon verifying the proper functioning of the organization in the simulation, the next step was to evaluate the behavior of the reallocation models that enable the dynamic adaptation of the organizational MAS. This was done through the distribution of the infrastructure resources in the CC platform among the different services offered in response to user demand.

5.1. Description of the initial state

The case study is based on a simulated *Denial of Service* (DoS) attack [34] using methods that expose the platform for persistence of files (FSS, File Storage Service). Concretely, the *GetSize* method is a complex function that uses recursion to calculate the sum of the size of the files contained in a directory.

The experiment presented below uses the same initial state as represented in Fig. 7, where the file storage service is deployed in different nodes (VM1 and VM2), each one hosted by a different physical machine (PR1 and PR2, respectively). As a result of this deployment, the service has a high availability (it is deployed in two servers), but it is at the same time located in physical machines with different computational loads, which is what occurs in a real environment, since both physical machines host other virtual machines that correspond to other services from the CC platform. In other words, the physical server PR1 has many available and unassigned resources, while PR2 has no available resources and the machines it hosts have a high computational load. This starting point, in addition to being didactic and easy to understand, reflects the typical deployment of any service in a CC environment. Likewise, Fig. 7 also shows the main agents that intervene in the readaptation process in the case study.

5.2. Development of the case study

In order to evaluate the readaptation of the infrastructure at a micro level, the first experiments are conducted during which an execution thread is progressively launched every second, up to a maximum of 10 s (10 threads). In this case, each thread continuously queries the GetSize web service from the file storage service. As previously indicated, this method is complex, since it performs recursive internal queries to find out the size of the object (file or directory). The QoS for the service is considered to be adequate when the response time to a query does not exceed 2.5 s.

The result of the experiment is presented in Fig. 8, which shows a graph with the response time for the GetSize method for the duration of the test. Once the system detects that the QoS level in the service has decreased, that is, when the average response time is greater than 2.5 s, it automatically initiates an adaptation process for the infrastructure at a micro level. As indicated in Fig. 8, once the auto-adapt process is complete and the value of the weights have been adjusted, we can see that the response time for the service returns to a value less than the acceptable QoS levels (less than 2.5 s).

We will now provide a detailed description of the process that was carried out during the distribution of infrastructure resources at a micro level. The algorithm begins when the *Service Monitor* agent detects that the QoS level associated with the service has reduced progressively, exceeding the acceptable minimum response time (2.5 s). The *Service Supervisor* then uses the QoS data for the service and decides to initiate the process for the redistribution of infrastructure resources at the micro level. The complete process is shown in Fig. 9.

The *Service Supervisor* agent first sends a message (Step 1, Fig. 9) to each of the *Local Manager* agents for each physical machine (PR1 and PR2) that hosts the worker nodes for the service in order to inform them that the FSS needs more resources.

This message will also indicate the weight for each node in the request balancing process. Each *Local Manager* agent uses this information independently to determine the amount of additional

[4] https://www.rabbitmq.com.
[5] https://libvirt.org/index.html.

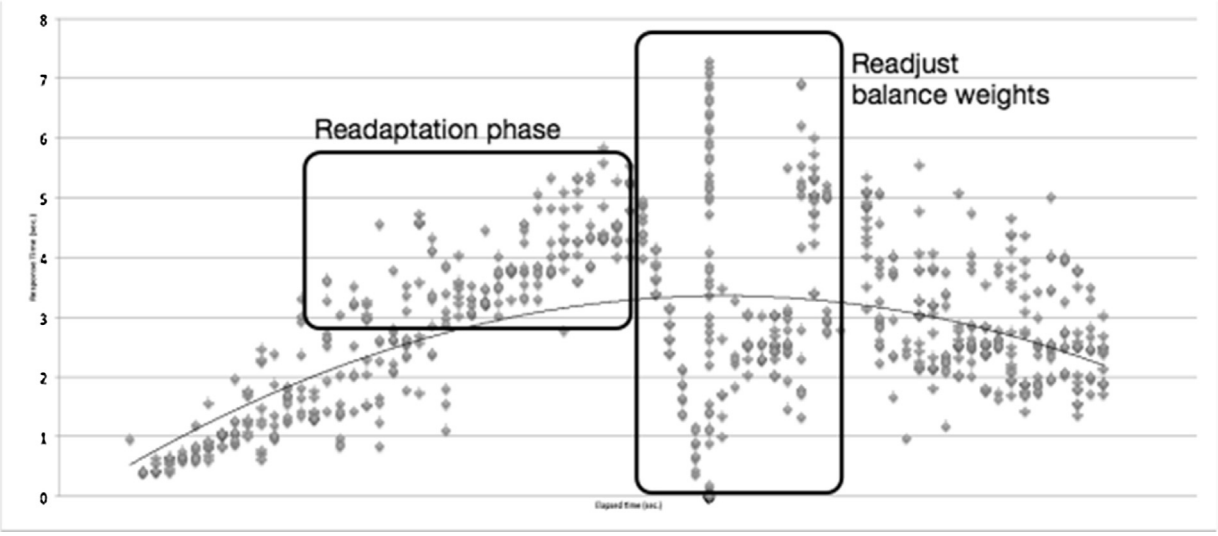

Fig. 8. Readjust the infrastructure resources at the micro level (GetSize method).

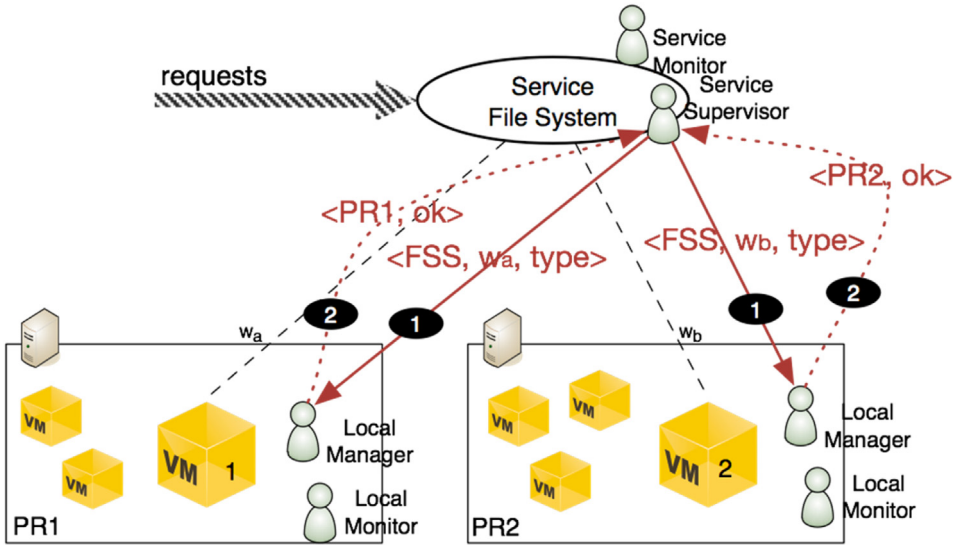

Fig. 9. Experiment 1: Exchange of messages in the distribution of infrastructure at the micro level.

resources that it can allocate to the service. To do so, the Local Manager agent for each machine asks its *Local Monitor* counterpart agent for information about the state of each machine; in other words, the instantiation I^{t}_{PR} that characterizes the physical equipment. The *Local Manager* agent retrieves all the local information that it uses to carry out the decision-making process. The next step is to determine how many resources (vCPU and memory) it can provide to the host node, which needs a greater amount of resources to handle existing demand. Once the reasoning process has finished, the two *Local Manager* agents located in PR1 and PR2 individually notify the *Service Supervisor* agent of the results of the provision of services process (PR1 provides more resources while PR2 does not).

Once the individual reasoning process for each physical server has finished, the two Local Manager agents located in PR1 and PR2 individually notify (Step 2, Fig. 9) the *Service Supervisor* of the results for the provision of services process. Once this information has been received, the *Service Supervisor* will not perform any additional action, since at this point at least one of the physical nodes (in this case both) has provided more resources for the service experiencing difficulties. In the meantime, the *Service Monitor* agent balances the weights to adjust demand to the ability of the nodes to provide the new resources. According to this new adjustment in the weights of the nodes, the *Service Supervisor* will not take any additional action if the average response time of the service decreases (less than the threshold of 2.5 s, as was the case).

The previously detailed experiment allowed us to analyze and validate the behavior of the MAS architecture and the proposed micro distribution algorithm in a specified performance test. In this experiment, the approach is applied to a method that requires high processing from the server side. As shown, the system fits properly, reducing response times to service difficulties. The algorithm allows a moderate allocation of resources for the computational resources that are assigned but not used, which suggests that it also maximizes the efficiency of the CC system as a whole. Furthermore, this method no longer requires an allocation of resources at a larger level (macro), which usually has a greater impact on the services provided by the CC platform.

In the same case study, similar to the previous experiment, additional experiments were conducted with other FSS methods. The

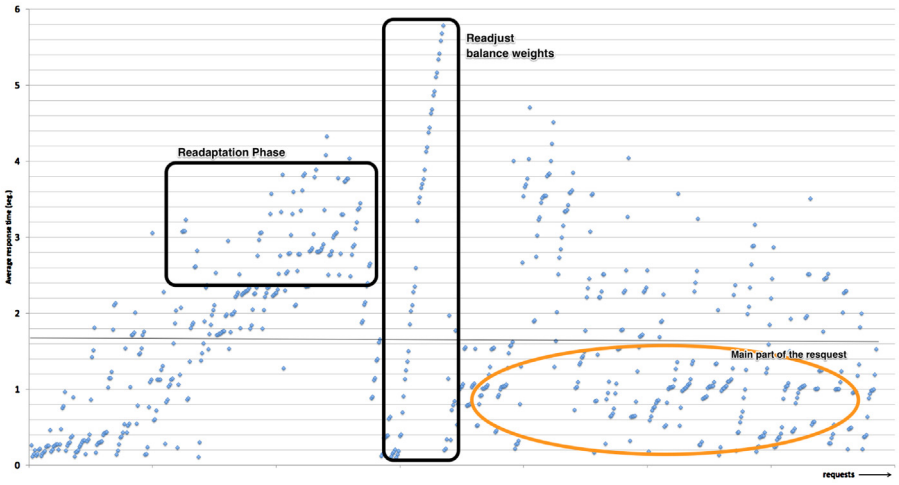

Fig. 10. Readjust the infrastructure resources at the micro level (GetSize method).

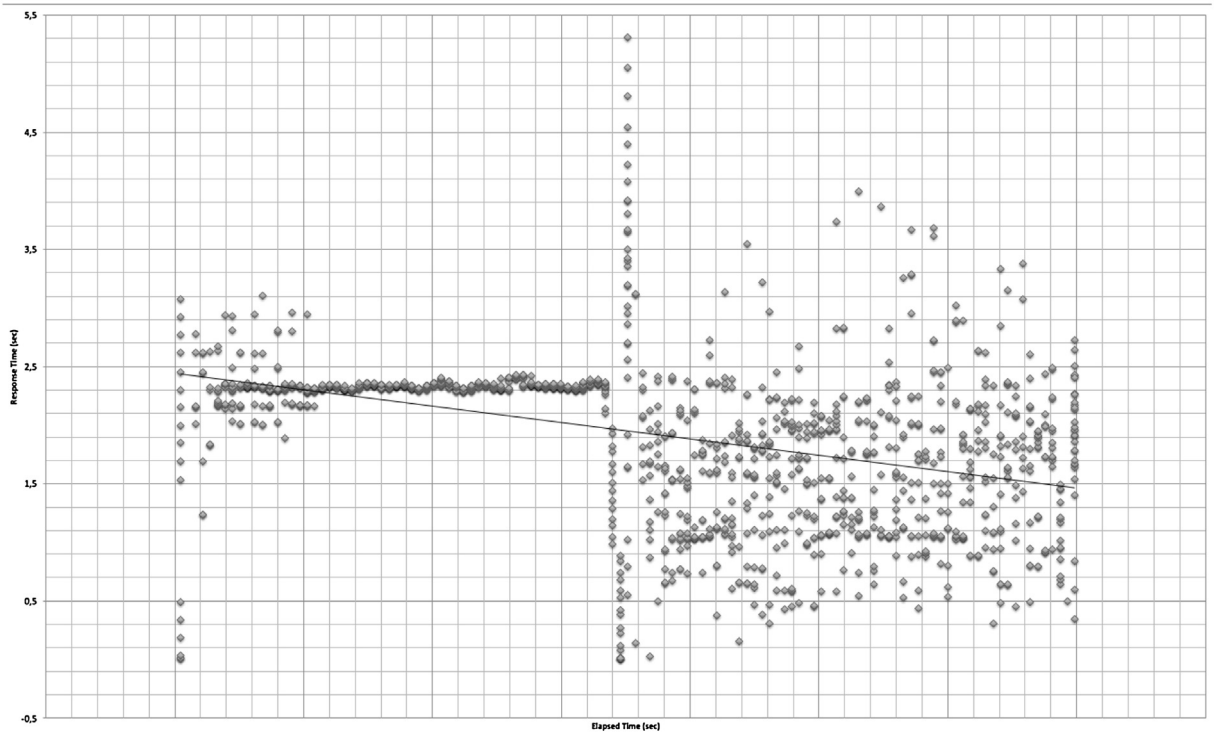

Fig. 11. Aggressive readjust the infrastructure resources at the micro level (GetSize method).

results were also satisfactory. For example, Fig. 10 shows how the algorithm behaves with *getFolderContent* method, which has low complexity. As shown, the response time for petitions at the beginning of the experiment increases considerably; however, once the adaptation is produced, the response times are reduced mostly staying within the limit (1.8 s). In this case, the reduction is not as great as with the previous case, since the characteristics of the function do not require computational power; therefore, in this case the bottleneck is at the persistence level of the information.

Finally, the last experiment is focused on an aggressive performance test, where the proposed algorithm is also able to adapt the CC resources and improve response times. In this experiment, the increase of requests is very aggressive at first, but subsequently remains constant, which results in a stable response rate, although not within the acceptable limit, as shown in Fig. 11. When the adaptation is reached by the MAS architecture, the average response time decreases significantly. However, this adaptation does not completely solve the problem, which would make this one case where the *Service Supervisor* agent should assess whether a macro allocation is needed.

5.3. Discussion

The previous section provided an empirical evaluation and validation in a case study of a MAS architecture and the adaptation models proposed within the framework of this research work. The experiments conducted have made it possible to validate various aspects as explained in detail below.

An empirical comparison of the proposed model with other existing approaches in the state of the art is not possible, since it is difficult to recreate the computational environments and/or simulation in which they were evaluated. However, it is possi-

ble to perform a theoretical comparison of the proposed approach with respect to other studies in the current state of the art. First, the proposed model follows a distributed approach to solve the problem, which is completely different from other studies in the state of the art [37,8]. This approach, which has been demonstrated to be valid for the distribution of computational resources in this type of CC environment, presents certain advantages with respect to availability since there is not just one component in charge of the distribution of resources; instead, it is the system itself that reorganizes through the individual adaptation of its components.

Furthermore, in the approaches in the current state of the art, the execution of the assignment algorithms is a complex task that requires a great deal of computational time and power [22]. In contrast, the proposed model simplifies the search for an appropriate solution to the problem because (i) it distributes the computational needs among different computational nodes. Moreover, the computational needs (represented by Local Manager and Local Monitor) are isolated within the resources associated with the physical server which can be configured. (ii) There are fewer values to consider since each node need only consider the data for its own resources. Finally, (iii) each node can autonomously apply a partial solution to the problem, thus eliminating the need to coordinate at the global level of the platform. In terms of the specific adaptation algorithms, the proposed solution uses optimization techniques, which have been previously used [27], but never following a distributed approach.

The other big difference in this research with regard to other approaches in the current state of the art is the minimum unit of distribution. While the minimum unit in the state of the art is usually the virtual machine [11,8], this research considers the minimum units to be the number of *vCPUs* and the virtual memory assigned to each virtual machine in the infrastructure. With this approach it is possible to distinguish the micro and the macro level in the reallocation of infrastructure resources, which makes it possible to solve the problem of demand without needing to instantiate virtual machines, which is in itself an energy efficient solution that maximizes the use of active computational resources while minimizing costs.

6. Conclusions

Thus, the architectural model developed in this research satisfies the objectives proposed at the beginning of the work. Specifically, this research validates that a VO-based architecture of MAS is able to act properly to monitor and control a CC environment. Also, a distributed RA algorithm was designed. The proposed approach for the allocation of resources is carried out on each node of the CC, and ultimately permits an improvement in the efficiency of the CC environment, minimizing the percentage of underused resources. Finally, both the MAS and the algorithm were tested in a real environment, making it easy to evolve both to a CC production.

However, we would propose the following lines of work that will be undertaken over a short and long term basis as a complement to the initially established objectives: (i) to extend the distribution algorithms for computational resources using two main objectives. First, we aim to adapt the proposed dynamic self-adapting model to then include all of the software layers of a CC system, including the persistence layer. And, (ii) to extend the proposed adaptation model to include other infrastructure products, especially those that allow high-performance computing centered on the massive analysis of data.

Acknowledgements

This work has been supported by the MICINN project TIN2012-36586-C03-03.

References

[1] H.A. Aikins, C.L. Darling, M.E. Gernaey, H.S. Kaldestad U.S. Patent No. 7, 296, 268. Washington, DC: U.S. Patent and Trademark Office, 2007.
[2] M. Alhamad, T. Dillon, E. Chang, Conceptual SLA framework for cloud computing, in: 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST), IEEE, 2010, pp. 606–610.
[3] P. Anedda, S. Leo, S. Manca, M. Gaggero, G. Zanetti, Suspending: migrating and resuming HPC virtual clusters, Future Gener. Comput. Syst. 26 (8) (2010) 1063–1072.
[4] F. Bahrpeyma, A. Zakerolhoseini, H. Haghighi, Using IDS fitted Q to develop a real-time adaptive controller for dynamic resource provisioning in Cloud's virtualized environment, Appl. Soft Comput. 26 (2015) 285–298.
[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, ACM SIGOPS Oper. Syst. Rev. 37 (5) (2003) 164–177.
[6] L.A. Barroso, The price of performance, Queue 3 (7) (2015) 48–53.
[7] L.A. Barroso, U. Hölzle, The datacenter as a computer: an introduction to the design of warehouse-scale machines, Synth. Lect. Comput. Archit. 4 (1) (2009) 1–108.
[8] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.
[9] A.B. Brown, N.A. Campbell, R.J. Cocks U.S. Patent No. 8, 959, 367. Washington, DC: U.S. Patent and Trademark Office, 2015.
[10] R. Buyya, High performance cluster computing: Architectures and systems (volume 1). Prentice Hall, Upper SaddleRiver, NJ, USA, 1, 1999, 999.
[11] R. Buyya, A. Beloglazov, J. Abawajy, Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges, in: Proceedings of the 2010 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA, July 12–15, 2010.
[12] R. Buyya, C.S. Yeo, S. Venugopal, Market-oriented cloud computing: vision, hype, and reality for delivering it services as computing utilities, in: 10th IEEE International Conference on High Performance Computing and Communications, HPCC'08, IEEE, 2008, pp. 5–13.
[13] S. Carlin, K. Curran, Cloud computing technologies, Int. J. Cloud Comput. Serv. Sci. 1 (2) (2012) 59–65.
[14] S. Caton, C. Haas, K. Chard, K. Bubendorfer, O.F. Rana, A social compute cloud: allocating and sharing infrastructure resources via social networks, IEEE Trans. Serv. Comput. 7 (3) (2014) 359–372.
[15] A. Chandak, K. Jaju, A. Kanfade, P. Lohiya, A. Joshi, Dynamic load balancing of virtual machines using QEMU-KVM, Int. J. Comput. Appl. 46 (2012) 10–14.
[16] D. Chiu, Elasticity in the cloud, ACM Crossroads 16 (3) (2010) 3–4.
[17] F. De la, S. Prieta, J. Rodríguez, J.M. Bajo, Corchado + Cloud: a virtual organization of multiagent system for resource allocation into a cloud computing environment, Trans. Comput. Collect. Intell. XV (2014) 164–181.
[18] F. De la Prieta, S. Rodríguez, J.M. Corchado, J. Bajo, Infrastructure to simulate intelligent agents in cloud environments, J. Intell. Fuzzy Syst. 28 (1) (2015) 29–41.
[19] I. Emeakaroha, M. Brandic, Low level Metrics to High level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments, International Conference on High Performance Computing & Simulation (2010) 48–54.
[20] P. Fisher, R. Pant, J. Edberg, Cloud Computing: Assessing Azure, Amazon EC2, Google App Engine and Hadoop for IT Decision Making and Developer Career Growth, ACM, Berkely, CA, USA, 2016.
[21] I. Goiri, F. Julià, J.O. Fitó, M. Macías, J. Guitart, Resource-level QoS Metric for CPU-based Guarantees in Cloud Providers. Economics of Grids, Clouds, Systems, and Services, Springer, Berlin Heidelberg, 2016, pp. 34–47.
[22] H. Goudarzi, M. Pedram, Multi-dimensional SLA-based resource allocation for multi-tier cloud computing systems, in: IEEE International Conference on Cloud Computing (CLOUD), IEEE, 2011, pp. 324–331.
[23] M.M. Hassan, M.S. Hossain, A.J. Sarkar, E.N. Huh, Cooperative game-based distributed resource allocation in horizontal dynamic cloud federation platform, Inf. Syst. Front. 16 (4) (2014) 523–542.
[24] S. Heras, F. De la Prieta, V. Julian, S. Rodríguez, V. Botti, J. Bajo, J.M. Corchado, Agreement technologies and their use in cloud computing environments, Progr. Artif. Intell. 1 (4) (2012) 277–290.
[25] D. Hutchins, Just in Time, Gower Publishing, Ltd., 1999.
[26] S. Kong, L. Feng, Dynamic performance management for database applications in virtualized environments, IEEE 3rd International Conference on Computer Research and Development (ICCRD) 2 (2011) 195–199.
[27] D. Kusic, J.O. Kephart, J.E. Hanson, N. Kandasamy, G. Jiang, Power and performance management of virtualized computing environments via lookahead control, Clust. Comput. 12 (1) (2009) 1–15.
[28] A. Li, X. Yang, S. Kandula, M. Zhang, CloudCmp: comparing public cloud providers. Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, ACM (2010) 1–14.
[29] W. Lin, C. Zhu, J. Li, B. Liu, H. Lian, Novel algorithms and equivalence optimisation for resource allocation in cloud computing, Int. J. Web Grid Serv. 11 (2) (2015) 193–210.
[30] J.Z. Luo, J.H. Jin, A.B. Song, F. Dong, Cloud computing: architecture and key technologies, J. China Inst. Commun. 32 (7) (2011) 3–21.

[31] M. Malhotra, R. Malhotra, Cloud adaptive resource allocation mechanism for efficient parallel processing, Int. J. Cloud Appl. Comput. 4 (4) (2014) 1–6.
[32] P. Mell, T. Grance, The NIST definition of cloud computing (draft), NIST 800 (145) (2011) 7 (Special publication).
[33] G. Nan, Z. Mao, M. Li, Y. Zhang, S. Gjessing, H. Wang, M. Guizani, Distributed resource allocation in cloud-based wireless multimedia social networks, Network 28 (4) (2014) 74–80.
[34] R.M. Needham, Denial of service, in: Proceedings of the 1st ACM Conference on Computer and Communications Security, ACM, (1993, December), 1993, pp. 151–153.
[35] P. Patel, A.H. Ranabahu, A.P. Sheth, Service Level Agreement in Cloud Computing, The Ohio Center of Excellence in Knowledge- Enabled Computing, 2016, pp. 2009.
[36] P.S. Pillai, S. Rao, Resource allocation in cloud computing using the uncertainty principle of game theory, IEEE Syst. J. (99) (2014) 1–12.
[37] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, X. Zhu, No power struggles: coordinated multi-level power management for the data center, ACM SIGARCH Comput. Archit. News 36 (1) (2008) 48–59.
[38] N. Regola, J.C. Ducom, Recommendations for virtualization technologies in high performance computing, in: 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2010, pp. 409–416.
[39] R.R. Schaller, Moore's law: past, present and future, Spectrum 34 (6) (1997) 52–59 (IEEE).
[40] S. Singh, I. Chana, Energy based efficient resource scheduling: a step towards green computing, Int. J. Energy Inf. Commun. 5 (2) (2014) 35–52.
[41] A. Singh, M. Malhotra, Agent based resource allocation mechanism focusing cost optimization in cloud computing, Int. J. Cloud Appl. Comput. (IJCAC) 5 (3) (2015) 53–61.
[42] V. Sriram, S. Ravimaran, Dynamic resource parallel processing and scheduling by using virtual machine in the cloud environment, Int. J. Eng. Res. Technol. 3 (4) (2014).
[43] D. Talia, Clouds meet agents: toward intelligent cloud services, IEEE Internet Comput. (2) (2012) 78–81.
[44] H.N. Van, F. Dang Tran, J.M. Menaud, Autonomic virtual resource management for service hosting platforms, Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing IEEE Computer Society (2009) 1–8.
[45] V. Vinothina, R. Sridaran, P. Ganapathi, A survey on resource allocation strategies in cloud computing, Int. J. Adv. Comput. Sci. Appl. 3 (6) (2012) 97–104.
[46] G. Von Laszewski, J. Diaz, F. Wang, G.C. Fox, Comparison of multiple cloud frameworks, in: IEEE 5th International Conference on Cloud Computing (CLOUD), IEEE, 2012, pp. 734–741.
[47] Y. Wang, X. Lin, M. Pedram, A game theoretic framework of SLA-based resource allocation for competitive cloud service providers, in: Sixth Annual Green Technologies Conference (GreenTech), IEEE, 2014, pp. 37–43.
[48] G. Wei, A.V. Vasilakos, Y. Zheng, N. Xiong, A game-theoretic method of fair resource allocation for cloud computing services, J. Supercomput. 54 (2) (2010) 252–269.
[49] X. Wen, G. Gu, Q. Li, Y. Gao, X. Zhang, Comparison of open-source cloud management platforms: openStack and OpenNebula, in: 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), IEEE, 2012, pp. 2457–2461.
[50] M. Wooldridge, N.R. Jennings, Intelligent agents: theory and practice, Knowl. Eng. Rev. 10 (2) (1995) 115–152.
[51] L. Wu, S.K. Garg, S. Versteeg, R. Buyya, SLA-based resource provisioning for hosted software-as-a-service applications in cloud computing environments, Trans. Serv. Comput. 7 (3) (2014) 465–485.
[52] Y. Oguchi, T. Yamamoto, Server virtualization technology and its latest trends, Fujitsu Sci. Tech. J. 44 (1) (2008) 46–52.
[53] X. You, X. Xu, J. Wan, D. Yu, Ras-m. Resource allocation strategy based on market mechanism in cloud computing, in: Fourth ChinaGrid Annual Conference, ChinaGrid'09. IEEE, 2009, pp. 256–263.