Heras Barberá, SM.; De La Piedra, F.; Julian Inglada, VJ.; Rodríguez, S.; Botti Navarro, VJ.; Bajo, J.; Corchado, JM. (2012). Agreement technologies and their use in cloud computing environments. Progress in Artificial Intelligence. 1(4):277-290. doi:10.1007/s13748-012-0031-9.

Progress in Artificial Intelligence manuscript No.
(will be inserted by the editor)

Agreement technologies and their use in cloud computing environments

Stella Heras · Fernando De la Prieta · Vicente Julian · Sara Rodríguez · Vicente Botti · Javier Bajo

Received: date / Accepted: date

Abstract Nowadays, cloud computing is revolutionizing the services provided through the Internet to adapt itself in order to keep the quality of its services. Recent research foresees the advent of a new discipline of agent-based cloud computing systems, that can make decisions about adaption in an uncertain environment. This paper discusses the role of argumentation in the next generation of agreement technologies and its use in cloud computing environments.

Keywords Cloud Computing · Distributed systems · Multi-Agent Systems · Virtual Organizations · Argumentation

1 Introduction

Nowadays, the cloud computing paradigm has emerged as a key component of the Future Internet. Concurrent research in the new area of cloud computing is putting an end on the everlasting problem of limited availability of computing resources. A cloud computing system must readjust its resources by taking into account the demand for its services. At technological level, the difficulties have been overcome in large part due to the use of virtualization of hardware resources [17]. However, how to assign the physical infrastructure among virtual machines is a current topic in some research fields [4]. This raises the need for designing protocols that provide the individual components of the cloud architecture with the ability to self-adapting and of reach agreements in order to deal with the changes in the demand of services.

In order to solve this problem, recent research has led to the advent of a new discipline of agent-based cloud computing systems for the future Internet [2]. The possibility of applying multi-agent systems (MAS), based on virtual organization (VO) leans to countless advantages due to the ability of these systems to adaption based on incomplete information of an open environment. Thus, the infrastructure resources will be managed in an elastic and intelligent way. Moreover recent developments in argumentation-based agreement models have provided the necessary technology to allow agents to engage in argumentation dialogues, harmonize beliefs, negotiate, collaboratively solve problems, etc [1]. In these argumentation frameworks agents are able to argue and reach agreements. The time has now come to bring the new developments of the Future Internet into the picture.

Conventional diagrams found on the Internet (and many computational models for reaching agreements) leave out the most numerous and important routers of all - people [3]. Taking systems of people and agents operating at a large scale into consideration offers an enormous potential for tackling complex applications and new business models. Therefore, in the next generation of agreement technologies, humans and agents should have the ability to establish a series of relationships, forming what might be called human-agent societies [19], and to reach agreements by using the unlimited

This work is supported by the Spanish government (MICINN), project reference: TIN2012-36586-C03-01

Stella Heras · Vicente Julian · Vicente Botti ·
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia (Spain)
E-mail: {sheras,vinglada,vbotti}@dsic.upv.es

Fernando De la Prieta · Sara Rodríguez · Juan M. Corchado · Javier Bajo
Department of Computer Science, University of Salamanca
Plaza de la Merced s/n, 37008, Salamanca (Spain)
E-mail: {fer,srg,jbajope}@usal.es

computational resources provided by cloud computing systems.

Mutual contributions in agreement technologies, multi-agent systems and cloud computing can advance research in these areas to the final establishment of the Future Internet.

This paper introduces the potential role of argumentation in the next generation of agreement technologies for cloud computing environments. Among the potential applications of argumentation in this area, the main focus here will be in the specific domain of the re-distribution of resources in cloud computing systems. Section 2 presents +Cloud, a new cloud computing system based on MAS. The section explains our argumentation and provides an example of operation. Then, Section 4 introduces related works and future challenges and summarises the results of this study.

2 Argumentation-based Approach for Resource re-Distribution in a Cloud Computing Environment

2.1 Preliminaries

As shown in Figure 1, a cloud computing environment can be analyzed from two points of view. From an external viewpoint, which is the perspective of the users (developers, manager, administrators, or end users), and from an internal viewpoint that includes all technical and deploy details about the computation environment.

- At the external level, a cloud computing system is composed of a set of services which are offered to the users. These services can be of three types: Software, Plataform and Infrastructure; commonly known as XaaS (X as a Service) [16]. The software and platform services can be considered as web applications with special features with regards to how they store their state or information. In this study, we use the term *service* to refer to each of these application. Infrastructure services, on the other hand, are not web applications, but they are a way of offering computational resources.
- At the internal level, the system consists of a set of physical machines (servers) which contribute to the system by means of their computational resources (processing capacity, volatile memory, etc.). Physical machines have basically two states: available or turn on or not available or turn off. The available physical machines host abstractions of hardware called virtual machines.
  Finally, it should be noted that it is not only necessary to have physical machines, but also the virtual machines as well have to be connected among them through an internal network. The features and topology of this network are not taken into account in this study.

A cloud computing environment by definition is a high availability system, which means that the quality of the services has to keep regular independently of their demand. In practice this means that each of the services at the PaaS or SaaS level have to deploy in n virtual machines. These virtual machines have to be distributed among m physical machines ($m \geq 2$). In this context, the services, which are deployed among several virtual machines, consume the resources of the own virtual machines (and hence, of the physical ones) depending on the demand at each moment. In order to maintain a quality level of the service, the number of virtual machines (and physical servers) must grow or decrease according to demand; which is referred to as elastic services.

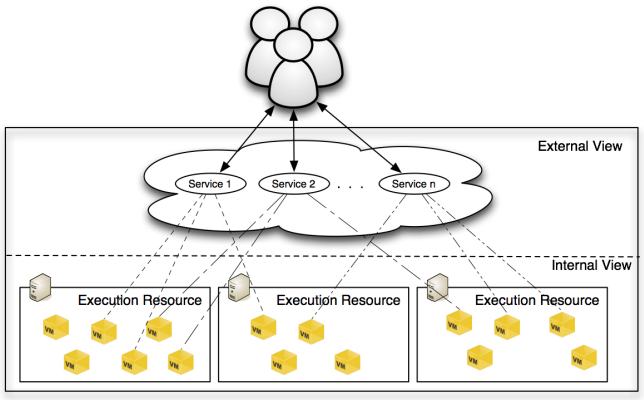

Fig. 1 External and internal view in a cloud computing environment

The latest generation of virtualization environments allows the resources available in the physical machines to be dynamically distributed among the virtual machines according to the current needs [17]. Therefore, it is not only possible to redistribute the resources of each physical machine among its virtual machines, but it is also possible to redistribute the resources at a global level among the entire cloud system. This means, for example that it is possible to switch on or switch off virtual machines in a specific physical server at any given moment, or even, to migrate virtual machines in execution between physical servers.

Nowadays, the greatest challenge in a cloud environment is how to efficiently re-distribute the available resources offered by physical machines among a variable set of virtual machines, taking into account the demand for the services offered by the system. In this section we

propose an argumentation-based approach to deal with the problem of resource re-distribution during a peak service demand in a cloud computing platform.

2.2 +Cloud Architecture

+Cloud is a platform based on the cloud computing paradigm. This platform allows services to be offered at the PaaS (Platform as a Service) and SaaS (Software as a Service) levels. The SaaS services are offered to the final users in terms of web applications, while the PaaS services are offered as web services. Both PaaS and SaaS layers are deployed using an IaaS (Infrastrucuture as a Service) layer, which provides a virtual hosting service with automatic scaling and functions for balancing workload. The IaaS layer is composed of the physical environment (the internal view) which allows the abstraction of resources shaped as virtual machines; however, the system does not offer this kind of service to the end users. A more detailed description of each layer is provided below:

SaaS Layer. This layer hosts a wide set of cloud applications. +Cloud as environment offers a set of native applications to manage the complete cloud environment: virtual desktop, user control panel and administration panel.
At this level, users have a personalized virtual desktop from which they can access their applications in the cloud environment, and other more general third party applications that use the services from the PaaS layer in order to save its state, enabling the elasticity of each application and the cloud environment in general.

PaaS Layer. The PaaS layer is oriented to offer services to the upper layer, and is supported by the lower IaaS layer. The PaaS layer provides services through RESTful web services [21] in an API format. One of the more notable services among the APIs is the identification of users and applications, a simple non-relational database service and a file storage area that controls versions and simulates a directory structure.The components of this layer are:
- the *IdentityManager*, which is the module of +Cloud in charge of offering authentication services to clients and applications;
- the *File Storage Service (FSS)*, which provides an interface for a container of files, emulating a directory structure in which the files are stored with a set of metadata, thus facilitating retrieval, indexing, search, etc; The implementation is based on the combination of several technologies such as GlusterFS [22] and MongoDB [23].
- and the *Object Storage Service (OSS)*, which provides a simple and flexible schemaless data base service oriented towards documents.

IaaS Layer. The objective of this layer is to offer this infrastructure service to upper layers of +Cloud (SaaS and PaaS). The key characteristic on a cloud environment is that the hardware layer includes the physical infrastructure layer and the virtual one (in terms of virtual machines). The virtual resources (number and performance of the processors, memory, disk space, etc.) are shown as unlimited, but they are supported by a limited number of physical resources, although the final user has the view of unlimited resources.

The virtual and physical resources are managed dynamically. To this end, a virtual organization of intelligent agents that monitor and manage the platform resources is used. This organization implements an argumentation-based agreement technology in order to achieve the distribution of resources depending on the needs of the whole system.

2.3 +Cloud Multiagent System.

The redistribution of resources can be seen from two points of view, from a micro level, and from a macro level. At the micro level, the system contains the redistribution of resources among virtual machines hosted within a single host. That is, a physical server has a number of available physical resources (processing, memory and disk) that have to be shared between different hosts virtual machines, leaving a minimal set of resources available to the host itself. At this level, the main objective is to maximize the use of resources, therefore, the intention is take out the underutilized resources of the system. Consequently, the physical machine and the hosted virtual machines will each make high use of physical resources available. At the macro level, however, the system contains the global redistribution of resources within the Cloud. This means that start or stop virtual machines on physical servers, or even migrate machines between physical servers, are available at all times. At the macro level, the goal is to minimize the use of resources, so that the largest number of machines remain disabled, but always maintaining the goal of high availability as a priority. Thus, power consumption and cooling requirements are reduced and the lifetime of physical machines is extended, which in turn makes it possible to reduce the maintenance cost of the cloud environment as well.

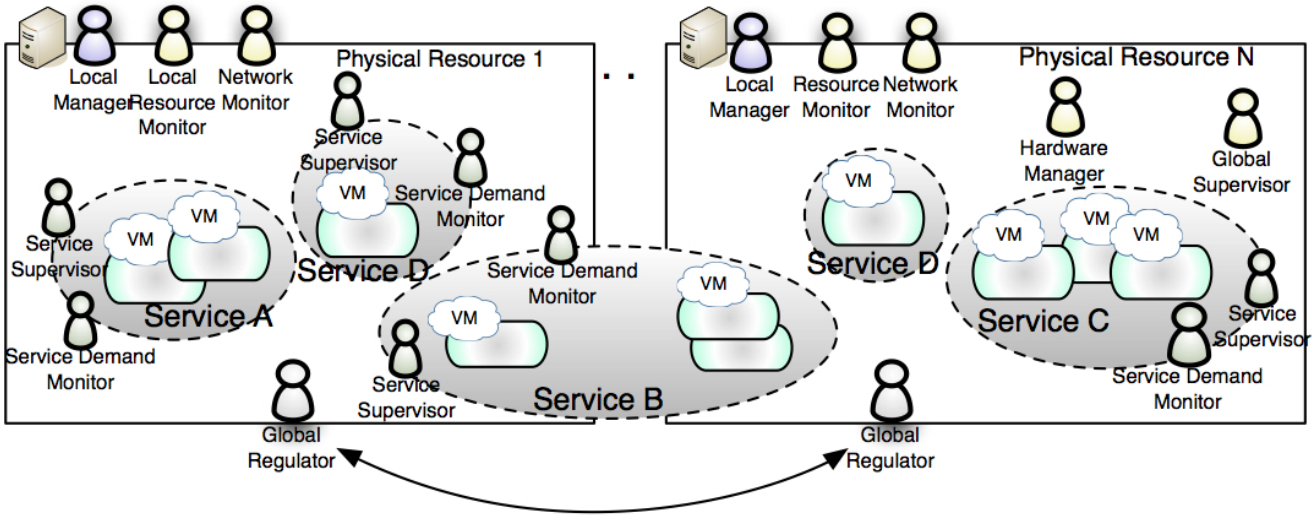

Fig. 2 Virtual Organizations for Elastic Management of Resources

In +Cloud, the elastic management of the available resources was done by a Multi-agent System (MAS) based on Virtual Organizations (VO), see Figure 1. With this approach it is more easier to design the overall system[18]. In the +Cloud VO there is a set of agents that are especially involved in the adaptation of the system resources in view of the changing demand of the offered services. Figure 2 presents the different roles that +Cloud agents can play. These are the following:

- *Service Demand Monitor (SDM)*, in charge of monitoring each demand service which is offered by +Cloud, which means service at SaaS and PaaS levels. There is one agent of this type per each kind of service. These agents will be able to offer information about not only the current demand, but also history of demands as well. They incorporate a load balancer to redirect the request to the different virtual machine which are offering the service at each time.
- *Service Supervisor (SS)*, this role is responsible for making decisions about each individual service. It receives information from the SDM of the same type of service, and is responsible for taking action if the quality level is not what is intended or if an error is occurring. There is one for each service and it is in the same virtual machine hosting the SDM of the same service, which in turn incorporates the load balancer service.
- Local Resource Monitor (LRM). in charge of knowing the usage level of the virtual resources for each virtual machine. There is one in each physical server and each one has all the knowledge about the physical resource and its virtual machine. However, it does not have any knowledge about other nodes in the cloud environment.
- *Local Manager (LM)*, in charge of allocating the resources of a single physical machine among its virtual machines and its own physical machine. There is one in each physical server and the knowledge is provided by the LRM of the same physical machine. It is able to redistributed the resources among the virtual machines based on its own knowledge, it also can turn on or shutdown a virtual machine of a give service. It can make autonomous decisions based on its knowledge.
- *Global Regulator (GR)*, in charge of negotiating the redistribution the resources at global level with its peers. This includes 1) Migration of virtual machines between physical servers; 2) Creation/Destruction of physical resources (turn on/turn off physical servers), as well as virtual resources (instantiating/ stopping virtual machines); or simply (3) redistribution of the resources in each physical server among its virtual resources. There is one in each physical server.
- *Global Supervisor (GS)*, which oversees and ensures that the other agents of the VO work correctly. In the event of something failing, or that one of the agents does not respond to its messages, this role should take the necessary actions to restore the system to a functioning state. Also, it may act as a judge in the event that GR that actors cannot agree on a particular service.

- *Hardware Manager (HM)*, the goal of this role is to manage the hardware that is in use and on standby at all times. It is able to start and stop hardware, as well as know the status for each system.
- *Network Monitor (NM)*, this role can monitor the network from the point of view of each single physical machine. This information allows the algorithm to make better decisions in an argument.

In this system, a peak in the demand of a specific service can give rise to an overload of one or more of the virtual machines that provide this service. Therefore, the system has to re-distribute its virtual and physical resources to cope with this problem. As indicated, resource redistribution can be done at the micro level and macro level; let us call them intra-machine or inter-machine, respectively.

Although the intra-machine level is not within the scope of this study, it is noteworthy that the redistribution of resources is made by the LM agent based on information provided by the LRM agent. Both agents are located in the physical machine. A model case-based reasoning that has a level of total knowledge is used to perform resource allocation.

When an LM agent detects that the physical server has insufficient resources to meet the demand for the service, or when an SS agent detects that the quality of service is not adequate, it is necessary to distribute to the macro level (inter-machine). It can also happen that the LM agent detects that the physical machine is underutilized, at which point it may try to offer its resources or even turn off the physical machine to save energy.

In these situations, the GR agents can implement several potential agreement processes in order to decide the best option for re-distributing physical resources among existing virtual machines in the overall cloud environment. Several types of solutions can be identified as potential outcomes of the agreement processes established:

- *Basic Solution*, consists of determining a set of physical machines that have to redistribute their internal resources to resolve the performance issues on their virtual machines.
- *Easy Solution*, consists of instantiating (starting) a new virtual machine of a particular service on a server that has sufficient resources. In most cases this also involves the Basic Solution in the machine that hosts the new virtual machine.
- *Complex Solution*, consists of migrating one or more virtual machines (running) between physical machines, resulting in a redistribution of global resources. This solution is only possible if the network load level is not high.
- *Expensive Solution*, consists of starting a new physical machine and migrating virtual machines from other physical machines to this new available server. This solution is only possible if the network load level is not high.
- *Half Solution*, consists of starting a new physics machine, and instantiating a new virtual machine for a particular service.
- *Cheap Solution*, consists of migrating all virtual machines from one physical server to other physical machines so that the physical machine can be turned off.
- *Cheap and Easy Solution*, consists of turning off the virtual machines from one physical server so that the physical machine can be turned off.

Any of these solutions entails an underlaying *negotiation* process to allocate virtual and physical resources among services to solve the overload problem. However, it is not our aim in this study to discuss the best negotiation mechanism to implement the solution itself, but to provide the agents of the system with the ability of engaging in an *argumentation* dialogue to collaboratively decide which would be the best solution to make *before* starting the negotiation. Our hypothesis is that agents may make the most of their experience and help each other to avoid complex negotiation processes that have a lower probability of ending in a successful allocation of resources in view of similar previous experiences. In this sense, our approach can be viewed as a model to guide the negotiation process and maximize its success.

3 Argumentation Model for Cloud Computing paradigm

In [1, Chapter 3] we presented a computational case-based argumentation framework that agents can use to engage in an agreement process to make a decision about a problem at hand. We assume that a problem can be characterised by a set of features that describe it. In this study, we apply this framework to model the argumentation dialogue among agents in the +Cloud platform.

This section summarises the main components of the framework, introduces the communication protocol that agents use to exchange their positions and arguments and finally, shows the preliminary evaluation results of the system presented in this work. In this system, agents are able to have an argumentation dialogue with their peers and learn from the experience by following a case-based reasoning process (i.e. saving

the knowledge acquired as cases in case-bases). Also, the agents of our framework use an argumentation protocol to manage positions and arguments during the argumentation dialogue. By this protocol, agents exchange ACL messages with specific locutions trying to reach an agreement about the best solution to apply for a specific problem. The full explanation about the reasoning process that agents use to generate, select and evaluate their positions and arguments is outside of the scope of this paper and can be found at [5]. Furthermore, the entire communication protocol is defined in [1, Chapter 4].

3.1 Argumentation Framework

Our case-based argumentation framework defines two types of knowledge resources that the agents can use to manage arguments (see Section 3.2 for specific examples in our application domain):

- **A case-base with domain-cases,** represents previous problems and their solutions. Agents can use this knowledge resource to generate their positions in a dialogue and arguments to support them by reusing the solutions applied to previous similar problems. Therefore, the *position* of an agent represents the *solution* that this agent proposes. Also, the acquisition of new domain-cases increases the knowledge of agents about the domain under discussion. The structure of domain-cases that an argumentation system that implements our framework has depends on the application domain.
- **A case-base with argument-cases,** stores the information about a previous argument that an agent posed in a certain step of a dialogue with other agents. Agents in our framework can use argument-cases 1) to generate new arguments; 2) to select the best position to put forward in view of past argumentation experiences; and 3) to store the new argumentation knowledge gained in each agreement process, improving the agents' argumentation skills. Their structure is generic and domain-independent.
- **A set of argumentation-schemes,** represents stereotyped patterns of reasoning [20]. Argumentation-schemes consists of a set of premises from which agents can draw specific conclusions. In this sense, argumentation-schemes represent general rules that hold in the domain under discussion (e.g. regarding exceptional situations that force agents to select an specific type of solution). In addition, argumentation-schemes include a set of *critical questions* that represent possible ways of attacking the conclusion drawn from the scheme (e.g. exceptions to the rule, other sources of information that invalidate the rule, etc.).

In our proposal, arguments that agents exchange are tuples of the form $Arg = \{\phi, v, < S >\}$, where ϕ is the conclusion of the argument, v is the value that the agent wants to promote and $< S >$ is a set of elements that justify the argument (the support set). Therefore, we follow the approach of *value-based argumentation frameworks* [6], which assume that arguments promote values and those values are the reason that an agent may have to prefer one type of argument to another. Values in this work can be considered as types of solutions. Then, an agent could prefer to promote the *quality* of solutions and, for instance, propose an *"EasySolution"* over a *"BasicSolution"*, since it knows by experience that the former type of solutions achieve more successful results, for instance, in re-distributing resources for a specific service. On the other hand, another agent could prefer to promote more *economic* solutions and, for instance, propose a *"BasicSolution"* that re-distributes the existing resources of a physical machine without incurring the cost associated with booting a new machine or starting a migration. Moreover, in our argumentation framework we take into account the preferences (*ValPref*) of each agent over the set of *values* pre-defined in the system to select among different arguments to propose. Furthermore, the *dependency relation* between the proponent's and the opponent's roles is also taken into account to evaluate arguments from other agents. In our framework, we consider three types of dependency relations (inherited from [7]):

1. *Power*, when an agent has to accept a request from another agent because of some pre-defined domination relationship between them (e.g. a hierarchy defined over roles);
2. *Autorisation*, when an agent has committed itself to another agent for a certain service and a request from the latter leads to an obligation when the conditions are met (e.g. agents in charge of virtual resources distributed across different physical machines that offer the same service); and
3. *Charity*, when an agent is willing to answer a request from another agent without being obliged to do so (e.g. an altruistic agent that selflessly shares its free resources).

The support set S can consist of different elements, depending on the argument purposed. For example, if the argument justifies a potential solution for a problem, the support set is the set of features (*premises*) that match the problem to solve, other extra premises that do not appear in the description of this problem

but that have been also considered to draw the conclusion of the argument, and optionally, any knowledge resource used by the proponent to generate the argument (*domain-cases*, *argument-cases* or *argumentation-schemes*). This type of argument is called a *support argument*. On the other hand, if the argument attacks the argument of an opponent, the support set can also include any of the allowed attack elements of our framework. These are *distinguishing premises*, *counter-examples*, or *critical questions*. A distinguishing premise is either a premise that does not appear in the description of the problem to solve and has different values for two cases or a premise that appears in the problem description and does not appear in one of the cases. A counter-example for a case is a previous case (i.e. a domain-case or an argument-case), where the problem description of the counter-example matches the current problem to solve and also subsumes the problem description of the case, but proposing a different solution. Also, as pointed out before, *critical questions* represent potential attacks that can defeat the conclusion of an argumentation-scheme. This other type of argument is called an *attack argument*.

The structure of domain-cases and the concrete set of argumentation-schemes that an argumentation system that implements our framework has depends on the application domain. However, the structure of argument-cases is generic and domain-independent (see Table 1 for an example). Argument-cases store the information about a previous argument that an agent posed in a specific step of a dialogue with other agents.

In argument-cases we store a *problem* description that has a *domain context* that consists of the *premises* that characterise the argument. In addition, if we want to store an argument and use it to generate a persuasive argument in the future, the features that characterise its *social context* must be kept as well. The social context of the argument-case includes information about the *proponent* and the *opponent* of the argument. Moreover, we also store the preferences (*ValPref*) of each agent or group over the set of *values* pre-defined in the system. Finally, the *dependency relation* between the proponent's and the opponent's roles is also stored.

In the *solution* part of argument-cases, we store the *conclusion* of the case, the *value* promoted, and the *acceptability status* of the argument at the end of the dialogue are stored. The last feature shows whether the argument was deemed *acceptable*, *unacceptable* or *undecided* in view of the other arguments that were put forward in the agreement process. Attacked arguments remain acceptable if the proponent of the argument is able to rebut the attack received, or if the opponent that put forward the attack withdraws it. This feature is used in the argument management process of our argumentation framework to represent the potentially high persuasive power of current arguments that are similar to previous arguments that were attacked but remained acceptable at the end of the agreement process (see [5] for a detailed explanation of this reasoning process).

Finally, the *justification* part of an argument-case stores the information about the knowledge resources that were used to generate the argument represented by the argument-case (the set of premises, domain-cases, argument-cases or argumentation-schemes). In addition, the justification of each argument-case has an associated *dialogue-graph* (or several), which represents the dialogue where the argument was put forward. In this way, the sequence of arguments that were put forward in a dialogue is represented, storing the complete conversation as a directed graph that links argument-cases. This graph can be used later to improve the efficiency of an argumentation dialogue, for instance, finishing a current dialogue that is very similar to a previous one that proposed a solution that ended up in an unsuccessful re-distribution of resources.

3.2 Example of argumentation Dialogue

In our framework, the protocol that agents use to exchange positions and arguments is represented by a set of locutions and a state machine that defines the allowed locutions that an agent can put forward in each step of the argumentation dialogue (presented in Figure 5). The transitions between states depend on the locutions that the agent can use in each step. The set of locutions of our argumentation protocol are the following:

- $open\_dialogue(a_s, q)$: an agent a_s opens the argumentation dialogue, asking other agents to collaborate to solve a problem q.
- $enter\_dialogue(a_s, q)$: an agent a_s engages in the argumentation dialogue to solve the problem q.
- $withdraw\_dialogue(a_s, q)$: an agent a_s leaves the argumentation dialogue to solve the problem q.
- $propose(a_s, p)$: an agent a_s puts forward the position p as its proposed solution to solve the problem under discussion in the argumentation dialogue.
- $why(a_s, a_r, \phi)$ (where ϕ can be a position p or an argument arg): an agent a_s challenges the position p or the argument arg of an agent a_r, asking it for a support argument.
- $no\_commit(a_s, p)$: an agent a_s withdraws its position p as a solution for the problem under discussion in the argumentation dialogue.

- $assert(a_s, a_r, arg)$: an agent a_s sends to an agent a_r an argument arg that supports its position.
- $accept(a_s, a_r, \phi)$ (where ϕ can be an argument arg or a position p): an agent a_s accepts the argument arg or the position p of an agent a_r.
- $attack(a_s, a_r, arg)$: an agent a_s challenges the argument arg of an agent a_r.
- $retract(a_s, a_r, arg)$: an agent a_s informs an agent a_r that it withdraws the argument arg that it put forward in a previous step of the dialogue.

In order to demonstrate how our argumentation framework can be applied to manage the service overload problem, this section will now describe the data-flow for the argumentation dialogue among several global regulators engaged in the agreement process.

The process starts when a service demand monitor notices an overload in a file storage service (FSS) that it controls. The service supervisor in charge of the virtual machines that offer this service then sends the load information about the resources associated to these virtual machines to the local manager of the physical machine that hosts them. After that, the local manager will analyse the demand of the service and ask the local resource monitor for information about the physical resources load. With all of this information, the local manager will make the best decision to re-distribute its virtual and physical resources to cope with this overload problem.

In this example, we assume that the local manager decides that the re-distribution of resources to deal with the peak of demand implies an inter-machine configuration. Therefore, it transfers the information to the service supervisor from which it received the overload problem. This service supervisor notifies the global regulator of the same physical machine, and then it starts an argumentation dialogue with its peers in other physical machines to decide which is the best solution to propose to its service overload.

We also assume that there are global regulators connected among them and that they are able to check the positions proposed by other global regulators. Furthermore, all agents are collaborative and follow the common objective of providing the best solution by making the most of their individual experiences. An agent that proposes a position, let us say a *proponent* agent, tries to persuade any potential *opponent* that has proposed a different position to change its mind.

In this example, we consider the following values that agents may want to promote:

- **Quality:** agents that promote this value will select those solutions that improve the quality of the service.
- **Economy:** agents that promote this value will select those solutions involving the lowest consumption of resources.

For purposes of clarity, all agents belong to the same group, although the scenario could be extended to consider agents that belong to different groups, representing organisations (for instance, those which group together agents that manage the same type of services). In addition, agents can play different roles (e.g. local manager, global regulator, etc.) and even act as representatives of a group (e.g in this agreement process, global regulators represent the other agents of their physical machine). Thus, this is a complex scenario that requires an argumentation framework that is able to take into account the social context of agents to properly manage the argumentation process.

In this setting, let us suppose that two agents that play the role of global regulators, GR2 and GR3, are arguing with the global regulator that started the agreement process, GR, to decide which is the best re-distribution of resources to deal with the FSS overload. The value preference order of GR1 promotes economy (EC) over quality (QU) (promotes saving resources over providing high quality solutions, QU<EC). Also, the example commands a dependency relation of charity (C) between two global regulators, except for the case of GR1, which has an authorisation dependency relation (A) over the global regulators GR2 and GR3, which allows it to ask them for resources to support the FSS service. GR2 prefers economy over quality (QU<EC) and GR3 prefers quality over economy (EC<QU). Also, all agents have their own knowledge resources (domain-cases case-base, argument-cases case-base and argumentation-schemes ontology to generate, select and evaluate positions and arguments).

The premises of the domain context would store data about the overloaded resource and other domain-dependent data about the current problem. For instance, the premises that characterise the problem q to solve are the following: service identifier ($p_1 = \{"Service" = FSS\}$), service current demand ($p_2 = \{"Demand" = SD1\}$), virtual machines associated ($p_3 = \{"VMs" = \{VM1, VM2\}\}$), physical resources associated to these virtual machines ($p_4 = \{"Resources" = \{VM1R, VM2R\}\}$), and resources usage ($p_5 = \{"ResourcesUsage" = \{VM1RU, VM2RU\}\}$).

In the first step of the argumentation process, GR1 will open the dialogue with its peers by conveying them the problem information to them with the locution *open_dialogue*. Then, global operators GR2 and GR3 will search their case-bases of domain-cases (DC2 and DC3 respectively) to generate their positions. In this case, the solution consists of a description, the solution

type and the value promoted with this solution. Figure 3 presents a potential domain-case that GR2 could retrieve to generate its recommended solution *"IR: Internal Re-distribution"* (since it has deemed it as similar enough to the current problem), which proposes redistributing resources inside the same physical machine, which is a *"BasicSolution"* solution that promotes *economy*. Note that in this example we assume that domain-cases also store the type of solution that they represent. Since GR2 has been able to generate a solution, it will use the locution *enter_dialogue* to engage in the argumentation dialogue and *propose* to present its solution.

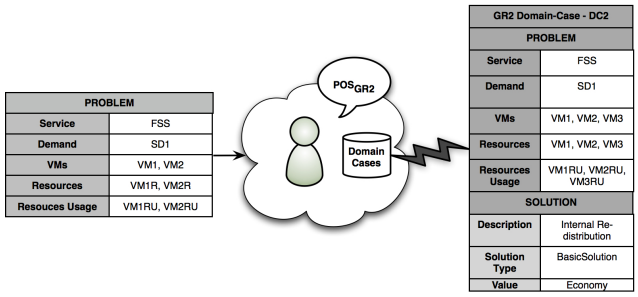

Fig. 3 Domain-Case DC2

In the case of GR3, Figure 4 shows that it has found the domain-case *DC3*, which proposes an alternative solution (e.g *"NVM: Instantiate a new VM"*) that promotes *quality* and has the type of solution *"EasySolution"*. Again, since GR3 has been able to generate a solution, it will use the locution *enter_dialogue* to engage in the argumentation dialogue and *propose* to present its solution.

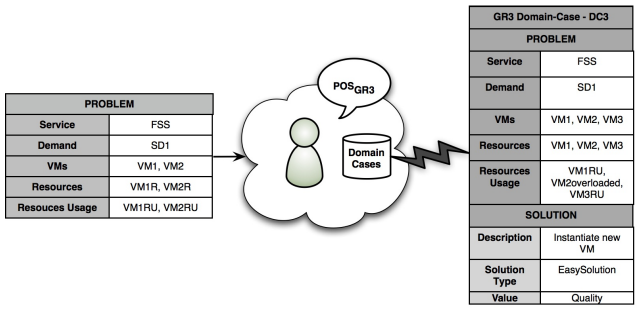

Fig. 4 Domain-Case DC3

Once the agents have proposed their positions, the GR1 has to decide which between them could be the best solution to apply. Therefore, it asks GR2 and GR3 to provide an argument for supporting their positions with the locution *why*. Assuming that GR2 and GR3 are willing to collaborate, they can put forward the following arguments by using the locution *assert*:

Support argument of GR2:
$SAGR2 = \{IR, EC, < Premises, \{DC2\} >\}$

Support argument of GR3:
$SAGR3 = \{NVM, QU, < Premises, \{DC3\} >\}$

where the support set includes the *premises* of the problem description and the domain-cases used by GR2 (DC2) and GR3 (DC3) to generate their positions.

DC2 and DC3 can be considered as counter-examples for each other (assuming that VM1overloaded subsumes the feature VM1RU pointing out a peak in the usage of this resource). As both GR2 and GR3 have a charity dependency relation between them, neither GR2 nor GR3 are committed by default to accept the argument of the other agent. Then, GR1 has to evaluate the arguments of GR2 and GR3 and decide between them. Now, let us suppose that GR! is receiving constant information about the resources load from its local manager (which in turn receives it from the local resource monitor). Then, let us suppose that GR1 knows an extra premise that states that there is a current overload in the virtual machine 1 (VM1Overloaded). This new premise matches an argumentation schemes of its ontology, S1, which changes its value preference order in case of any overload in a virtual machine (inspired by Waltons's *argument for an exceptional case* [20]):

Major Premise: if the case of x is an exception, then the value preference order can be waived and changed by EC<QU in the case of x.
Minor Premise: the case of overload is an exception.
Conclusion: therefore the value preference order can be waived and changed by EC<QU in the case of network overload.

Thus, this scheme will change the social context of the attack argument that the GR1 is going to create. As the support set of $SAGR2$ and $SAGR3$ contains a domain-case, GR1 will try to propose a counter-example or a distinguishing premise for these cases.

Thus, GR1 will check its case-base of domain-cases to find counter-examples for DC2 and DC3. Suppose that GR1 finds one counter-example for each case (DC1a for DC2 and DC1b for DC3) which subsume the problem description of each of these cases (but also including the new premise that states the overload of VM1), but providing opposite solutions (e.g. "Instantiate a new VM" for DC1a and "Internal Re-distribution" for DC1b). It could, therefore, generate the following attack arguments:

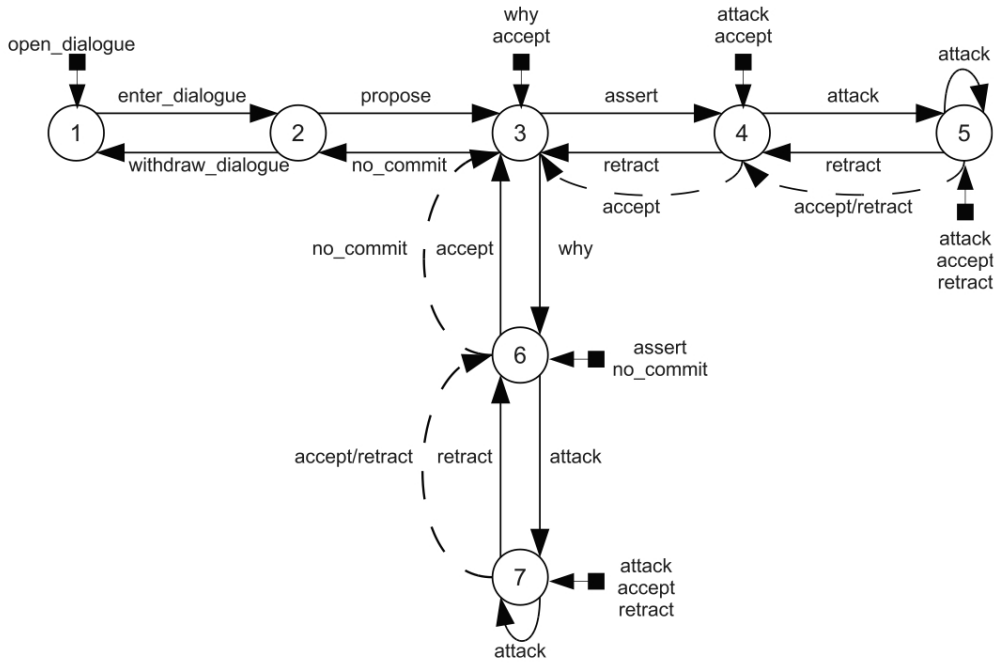

Fig. 5 Argumentation state machine of the agents

AA1 = {NVM, QU, <Premises ∪ {VM1Overloaded}, S1, {DC1a}>}

to undercut SAGR2 by attacking its support element DC2 with the counter-example DC1a. Here we assume that by attacking the argument of GR2, GR1 supports the argument of GR3 and then promotes quality (QU).

AA2 = {IR, EC, <Premises ∪ {VM1Overloaded}, {DC1b}>}

to undercut SAGR3 by attacking its support element DC3 with the counter-example DC1b. Here we assume that by attacking the argument of GR3, GR1 supports the argument of GR2 and then promotes economy (EC).

GR1 will then try to find distinguishing premises and will check that the problem description of domain-cases DC2 and DC3 matches the extended description of the problem (the original description plus the new premise VM1overloaded). Then, GR1 realizes that DC2 does not match with the extended description and generates an attack argument to GR2:

AA3 = {NVM, QU, <Premises ∪ {VM1overloaded}, {VM1overloaded}>}

to undercut SAGR2 by attacking its support element DC2 with the distinguishing premise *VM1overloaded*. Again, we assume that attacking the argument of GR2, GR1 supports the argument of GR3 and then promotes quality (QU).

Now, GR1 has to select the argument that it will pose to attack the positions of the other global regulators. Note that, if we assume that agents always observe their value preference orders when putting forward arguments, GR1 would prefer to pose AA1 and AA3 first than AA2 (since GR1 has the new value preference order changed to EC<QU by the argumentation-scheme S1). However, it has still to decide which AA1 or AA3 it would select to attack SAGR2. To do that, it checks its argument-cases case-base to decide which is the best argument to pose in view of its previous argumentation experience. Now, let us suppose that GR1 finds a similar argument-case for AA3 that was unaccepted at the end of a dialogue where it was also in the exceptional situation of an overload in the VM1, shown in Table 1. However, a counter-example that also included the extra premise, let us say DC4, was able to defeat the argument represented by the argument-case. Therefore, GR1 can infer that GR2 has enough pieces of evidence to defeat the distinguishing premise attack AA3 at the end of the dialogue. Thus, GR1 will use the locution *attack* and put forward AA1 to attack the position of GR2.

When GR2 receives the attack, it has to evaluate the attack argument in view of its support argument. Then, it will realise that SAGR2 does not defeat AA1 from its point of view, since GR1 has an authorisation dependency relation with it. Then, it would try to

PROBLEM	Domain Context	Premises ∪ {VM1overloaded}	
	Social Context	Proponent	ID = GR1
			Role = Global Regulator
			ValPref = EC<QU
		Opponent	ID = GR2
			Role = Global Regulator
			ValPref = QU<EC
		Dependency Relation = Authorisation	
SOLUTION	Conclusion = "Instantiate new VM"		
	Value = QU		
	Acceptability Status = Unaccepted		
	Received Attacks	Distinguishing Premises = ∅	
		Counter Examples = DC4	
JUSTIFICATION	Cases = ...		
	Dialogue Graph = ...		

Table 1 Argument-case representing SA1

generate more support for its position. In case that it cannot find more support, GR2 would have to use the *no_commit* locution to withdraw its position pos_GR2. If no more positions or arguments are generated, GR3 solution would be selected as the best to deal with the overload problem of service FSS and GR1 will send it an *accept* locution for its position. In this example, it is not necessary for the Service Supervisor to validate the solution because the high availability of the service is ensured.

The dialogue finishes when no new positions or arguments are proposed after a certain time. Then, the global regulator that initiated the agreement process retrieves the active positions of the participants, and the most accepted position (if several remain undefeated) is selected as the final solution to propose. In case of a draw, the final solution will be the most frequent position generated by the global regulators during the argumentation dialogue. Finally, once a position is selected as the outcome of the agreement process, the global regulator sends it to the local manager of its physical machine and both would start the process to implement it (with further negotiations if necessary). Also, at the end of the argumentation dialogue, all agents update their domain-cases case-bases with the new problem solved and their argument-cases case-bases with the information about the arguments proposed, with the attacks received, the final acceptability state, etc.

3.3 Preliminary Evaluation of the argumentation framework

With this scenario we have demonstrated how agents' arguments can be computationally managed in the proposed argumentation framework. The example shows the way in which agents can use the knowledge resources of the framework to generate, select and evaluate positions and arguments. Also, it takes into account the social context of agents to perform these activities. Our argumentation framework has also been implemented and tested to enhance a real customer support application run by a Spanish company [5].

Let us show a real example where the argumentation framework solves an overload issue and, therefore, a lost in the quality of service offered by the +Cloud environment. The test has been performed using a control enviroment with high heterogeneous physical machines (HP, Mac Server, Dell, etc.) with Centos linux operating system (https://www.centos.org/), the resource of these server are also very heterogeneous in terms of CPU, memory, storage, even the system architecture is disparate.

Following the previous example (Section 3.2), where the system avoids an overload problem due to an increment in demand; the FSS is going to be subjected to a stress test where the system will have to adapt itself. During this test, the argumentation framework will be key, since it will making the best decision in order to solve the overload problem.

The FSS characterization during the test will be the following:

- The FSS will be deployed in two virtual machines (VM1 and VM2)
- Each virtual machine will be deployed on a different physical server (VM1R y VM2R).
- VM1R and VM2R host an unknown number of virtual machines, and all the physical resources will be shared among them. We assume that the redistribution of the resources at the intra-machine level is not possible.
- The load balancer of FSS is allocated in VM1.
- The cloud environment will have an unknown number of physical servers.

The test consists of making a set of calls to a specific heavy web service exposed by FSS. This test is designed so that the load of the service increments lineally. There will be a set of threads that will be progressively launched, this means that at the beginning of the test, there will be just one thread and periodically one new thread will be launched (each 3 seconds), up to a maximum of 40 threads in parallel. The duration of the test is about 130 seconds. Each thread, continuously sends requests to the specified service of the FSS. This

specific method is called *GetFolderContents* which returns a list of folders and files of a specific path. In this case, the specific path is the root system ('/'). In this method, there is a special set of files and folders prepared for the test. The response time varies between 0,5 and 4 seconds.

The vertical axis, in Figure 6 represents the response time in seconds of each call to the method.

The above graph shows the process of re-distributing the resources within the system. This process includes the following steps:

1. The first part of the graph shows the linear increment in the number of requests. The tendency in this part of the graph has a pronounced slope. See *Tendency before adaptation* on Figure 6.
2. Once the service monitor detects that the quality of service is decreasing (since there is a big number of requests and the response time is bigger than 4 seconds), it notifies the problem to the service supervisor of the FSS.
3. The service supervisor notifies the problem to the local managers of VM1R and VM2R, but they cannot perform an intra-machine re-distribution of resources.
4. The service supervisor notifies the problem to the global regulator of its physical machine.
5. The global regulator initiates an argumentation with other global regulators, which are in other physical machines. Although this argumentation is complex, it does not give rise to a delay in the system. The result of the argumentation is to instantiate a new virtual machine in another server. See *Argumentation Phase* on Figure 6.
6. The next step is to adapt the system, this means turning on a new VM in another server (VM3R) and adding it to the FSS. As we can see in Figure 6 the load balancer retains the majority of the requests temporaly and adding it the new VM to the FSS. When the process is finished the load balancer redirects the retained requests to the VM that offers the service.
7. After the adaptation of the system, the response time does not increase. It is necessary to take into account that at the final part of the experiment the number of queries is greater because there are more threads running (up to 40).
8. Finally, it should be noted that the global tendency of the response time increases but the adaptation makes it possible to reduce the increase.

In the fifth step of the process, the system global regulator has an argumentation dialogue with its peers. At the beginning of this argumentation the global regulator builds the problem, q, which is composed of the following elements:

- $p_1 = \{"Service" = FSS\}$, the identification of the service, in this case FSS.
- $p_2 = \{"Demand" = (response\_time = 4,1), (requests = 94), (quality = 0,65)\}$, which is the actual demand of the service represented by the average response time, the number of requests and the calculated percentage of quality, respectively.
- $p_3 = \{"VMs" = \{VM1, VM2\}\}$ with the current virtual machines of the services, the characterization of each virtual machine is based on the vector $< V_{MV} >$ which includes information about the processor, memory, hard disk, kind of service, band wide, etc.
- $p_4 = \{"Resources" = \{VM1R, VM2R\}\}$ with the current physical machines of the service, the characterization of each physical machine is also based on a vector $< V_{PM} >$ but it additionally includes the free resources and an indicator of the objective performance of the physical server.
- $p_5 = \{"ResourcesUsage" = \{VM1RU, VM2RU\}\}$ with the current use of resources.

Then the problem q the argumentation dialog continuous as described in the before section 3.2.

4 Discussion

Recent research foresees the advent of a new discipline of agent-based cloud computing systems on the Future Internet. This paper identifies research opportunities from the Multi-Agent Systems (MAS) technology to the cloud computing paradigm and vice-versa. Agents are intelligent, flexible, autonomous and proactive; all of these features are needed by a cloud computing system where the resources have to be re-distributed within the system in order to cope with the demand.Current literature, reflects few references of studies that combine both agents and cloud computing paradigms: in [11] software agents appear as a new cloud computing service which would represent clients in virtual environments; [12] presents a complex cloud negotiation mechanism that supports negotiation activities in interrelated markets; and [14] presents a service-oriented QOS-assured cloud computing architecture. These contributions pave the way for an interesting new area of investigation in cloud-based multi-agent systems.

Argumentation-based agreement technologies, as a proficient research area within MAS-based agreement models, should echo these opportunities and contribute toward the achievement of new challenges in agent-based cloud computing. In recent years, the commu-

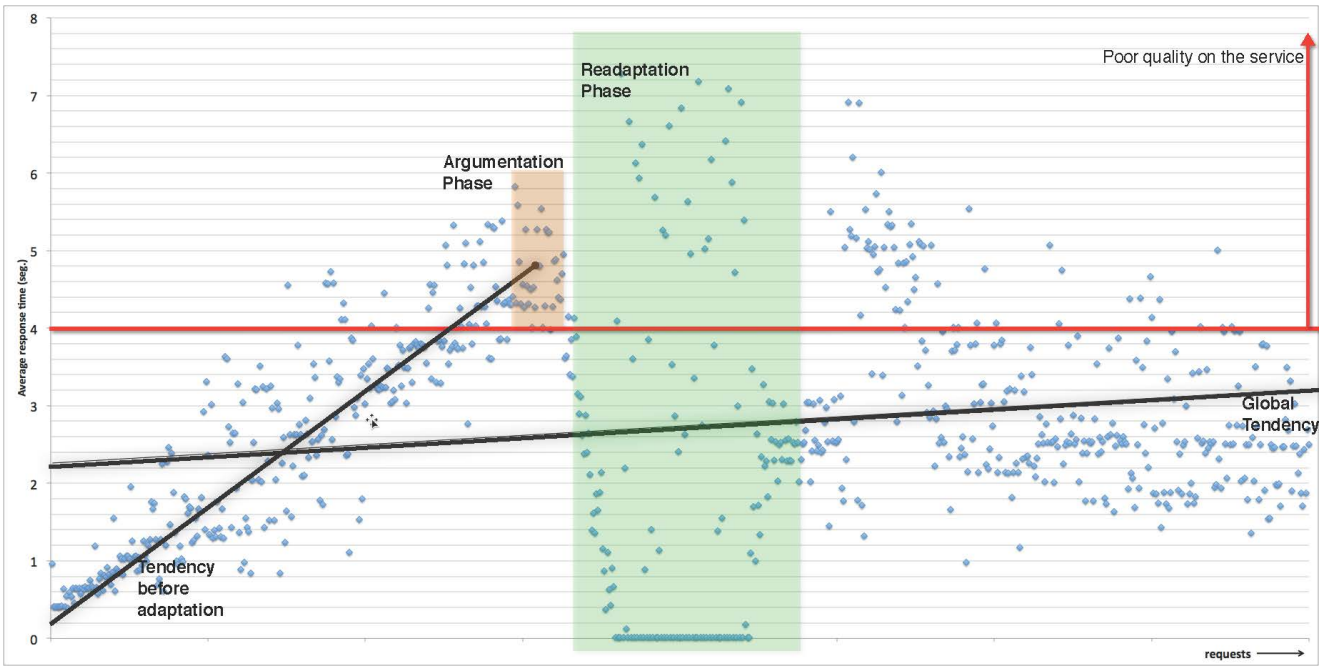

Fig. 6 Results of stress test

nity of argumentation in MAS has advanced research in many fields in the area of applying argumentation theory to harmonise incoherent beliefs among agents and to model the interaction among a set of agents [15]; however, but the application of argumentation approaches to cloud computing is a new challenge. Specifically, we have used an argumentation-based approach to reach an agreement about the best solution to implement for the re-distribution of resources when facing a peak service demand. To the best of our knowledge, we have presented the first argumentation-based solution for load-balancing services based on MAS cooperation. Thus, we deal with one of the main challenges for agent-based solutions to clouds software infrastructure, which states the advantages of using agents to create intelligent and flexible cloud services [2]. These include Service Level Agreements (SLAs) based on negotiation agents and load-balancing services based on MAS cooperation.

In doing so, we propose the first (to the best of our knowledge) argumentation-based solution for load-balancing services based on MAS cooperation (one of the open issues identified in [2]). Work is currently underway the implement and test this system, in order to analyze the viability and advantages of this approach. Also, the advantages that this approach contributes over direct resource allocation algorithms must be analyzed.

With regards on further work, on the one hand, in argumentation in cloud computing can elicit more argumentation-based agreement models that enable agents to argue and meet their goals within a society. Some application examples may include: negotiating Service Level Agreements; providing a method to harmonize conflicts that arise in the adaption of the system to environmental changes; and enabling a collaborative deliberation to find the best alternative for service composition. On the other hand, in cloud computing paradigm the future work will be focus on perform new test on different environments (number of servers, demand of the services, communication issues, etc.), as well as, the improve of the current algorithms.

References

1. Heras, S.: Case-Based Argumentation Framework for Agent Societies. PhD thesis, Universitat Politècnica de València. http://hdl.handle.net/10251/12497 (2011)
2. Talia, D.: Clouds meet agents: Toward intelligent cloud services. IEEE Internet Computing **16**(2) (2012) 78–81
3. Ashton, K.: That 'internet of things' thing. RFID Journal (2009)
4. Wang, L., et. al.: Scientific cloud computing: Early definition and experience. In: 10th IEEE Int. Conf. on High Performance Computing and Communications (HPCC-08), IEEE Press (2008) 825–830
5. Heras, S., Jordán, J., Botti, V., Julián, V.: Argue to agree: a case-based argumentation approach. International Journal of Approximate Reasoning (In Press)
6. Bench-Capon, T., Sartor, G.: A model of legal reasoning with cases incorporating theories and values. Artificial Intelligence **150**(1-2) (2003) 97–143

7. Dignum, F., Weigand, H.: Communication and Deontic Logic. In: Information Systems Correctness and Reusability, World Scientific Pub. Co. (1995) 242–260
8. Jordán, J., et. al.: A customer support application using argumentation in multi-agent systems. In: 14th Int. Conf. on Information Fusion. (2011) 772–778
9. The future of cloud computing. Technical report, European Comission (2010)
10. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review **10**(2) (1995) 115–152
11. Lopez-Rodriguez, I., Hernandez-Tejera, M.: Software agents as cloud computing services. In: 9th Int. Conf. on Practical Applications of Agents and Multiagent Systems. Volume 88 of Advances in Intelligent and Soft Computing., Springer (2011) 271–276
12. Sim, K.M.: Towards complex negotiation for cloud economy. In: 5th Int. Conf. on Advances in Grid and Pervasive Computing. Volume 6104 of LNCS., Springer (2010) 395–406
13. Aversa, R., et. al.: Cloud agency: A mobile agent based cloud system. In: Int. Conf. on Complex, Intelligent and Software Intensive Systems, IEEE Computer Society Press (2010) 132–137
14. Cao, B.Q., et. al.: A service-oriented qos-assured and multi-agent cloud computing architecture. In: 1st Int. Conf. on Cloud Computing. Volume 5931 of LNCS., Springer (2009) 644–649
15. Rahwan, I., Simari, G., eds.: Argumentation in Artificial Intelligence. Springer (2009)
16. Schaffer, H.E.: X as a Service, Cloud Computing, and the Need for Good Judgment IT Professional , vol.11, no.5, pp.4-5, 2009 DOI: 10.1109/MITP.2009.112
17. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. : Xen and the art of virtualization. In SOSP03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 164-177, New York, NY, USA, 2003. ACM.
18. Ruben Fuentes-Fernandez, Samer Hassan, Juan Pavon, Jose Manuel Galan, Adolfo Lopez-Paredes: Metamodels for role-driven agent-based modelling. Computational & Mathematical Organization Theory 18(1): 91-112 (2012)
19. Klusch, M.: Information agent technology for the Internet: A survey Data & Knowledge Engineering, Volume 36, Issue 3, March 2011, Pages 337-372.
20. Walton, D., Reed, C., Macagno, F.: Argumentation Schemes. Cambridge University Press (2008)
21. Richardson, L and Ruby, S.: RESTful Web Services, Web services for the real world O'Reilly Media, May 2007, Pages: 454
22. GlusterFS Developers. http://www.gluster.org The Gluster web site, 2012
23. Chodorow, K. and Dirolf, M. MongoDB: The definitive Guide. O'Reilly Media, Inc., 2010

Acronym	**Description**
A	Autorisation (agreement dependency relation)
C	Charity (agreement dependency relation)
EC	Economy (agreement objective)
DC	Domain Case
FSS	File System Storage (PaaS module)
GR	Global Regulator (Role)
GS	Global Supervisor (Role)
HM	Hardware Manager (Role)
IaaS	Infrastructure as a Service
LM	Local Manager (Role)
LRM	Local Resource Monitor (Role)
MAS	MultiagentSystem as a Service
NM	Network Monitor (Role)
OSS	Object System Storage (PaaS module)
P	Power (agreement dependency relation)
PaaS	Platform as a Service
QU	Quality (agreement objective)
RU	Resource Unity
S	Support set (agreement element)
SA	Support Argument
SaaS	Software as a Service
SD	Service Demand
SDM	Service Demand Monitor (Role)
SS	Service Supervisor (Role)
VM	Virtual Machine
VO	Virtual Organizations
XaaS	Something as a Service