The Role of Argumentation on the Future Internet: Reaching agreements on Clouds*

Stella Heras[1], Fernando de la Prieta[2], Sara Rodríguez[2], Javier Bajo[2], Vicente Botti[1], and Vicente Julián[1]

[1] Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València
Camino de Vera s/n, 46022 Valencia (Spain)
{sheras,vbotti,vinglada}@dsic.upv.es
[2] Department of Computer Science, University of Salamanca
Plaza de la Merced s/n, 37008, Salamanca (Spain)
{fer,srg,jbajope}@usal.es

Abstract. Recent research foresees the advent of a new discipline of agent-based cloud computing systems on the Future Internet, which would provide processing and memory resources for agents and intelligent cloud services at unprecedented scale. In this paper we discuss the role of argumentation on the next generation of agreement technologies in cloud computing environments and provide an example application.

Keywords: Cloud Computing, Argumentation, Multi-Agent Systems

1 Introduction

Recent developments on argumentation-based agreement models have provided the necessary technology to allow agents to engage in argumentation dialogues and harmonize beliefs, negotiate, collaboratively solve problems, etc [1]. However, these models follow a client-server approach, where computing and storage capacities depend on the static set of (possibly distributed) machines where the system is deployed. Therefore, this approach raises critical shortcomings in terms of performance, reliability, scalability or security. Fortunately, concurrent research on the new area of cloud computing is putting an end on the everlasting problem of limited availability of computing resources. Now, the cloud computing paradigm has emerged as a key component of the Future Internet.

Once we have client-server systems where agents are able to argue and reach agreements, now is time to put the new developments of the Future Internet into the picture. Recent research foresees the advent of a new discipline of agent-based cloud computing systems on the Future Internet, which would provide processing and memory resources for agents and intelligent cloud services at unprecedented scale [2]. In this context, the infrastructure resources will be managed following

* AT2012, 15-16 October 2012, Dubrovnik, Croatia.

an elastic and intelligent way. In addition, conventional diagrams of the Internet (and many computational models for reaching agreements) leave out the most numerous and important routers of all - people [3]. Considering systems of people and agents operating at a large scale offers an enormous potential for tackling complex applications and new business models. Therefore, on the next generation of agreement technologies humans and agents should have the ability of establishing a series of relationships, forming what might be called human-agent societies, and reach agreements by using the unlimited computational resources provided by cloud computing systems. As part of these technologies, argumentation-based agreement models would enable agents to argue and meet their individual or collective goals within a social structure.

On the other side, an environment based on cloud computing must readjust its resources taking into account the demand of its services. At the technological level, the difficulties have been overcome thanks to the use of virtualization of hardware resources [4]. However, how to assign the physical infrastructure among the virtual machines is a current topic in some research fields [5]. In this sense, there is a need for knowing what physical server hosts each virtual machine and its level of resources (processor and memory). In order to make these decisions, the hardware management components have a limited information based on the observation of the environment and its own experience. Moreover, sometimes it is necessary to readjust the hardware resources of several servers at the same time. This raises the need for designing protocols that provide the individual components of the cloud architecture with the ability of adapting itself and of reaching agreements in order to deal with the changes in the service demand.

Mutual contributions in agreement technologies and cloud computing can advance research in both areas to the final establishment of the Future Internet. In this paper we introduce the potential role of argumentation on the next generation of agreement technologies in cloud computing environments. Among the potential applications of argumentation in this area, we focus here on the concrete domain of resources re-distribution in cloud computing systems. Section 2 develops our proposal. Then, Section 3 introduces related work and future challenges and summarises the results of this work.

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

A current open issue in a cloud computing system is how to efficiently re-distribute resources among a variable set of virtual machines, considering 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. To illustrate our proposal, we provide an example where a set of agents in charge of managing virtual machines and physical resources in the platform engage in an argumentation dialogue to reach an agreement about the best solution to make when a service is failing due to an overload in the virtual machines that provide this ser-

vice. First, the platform and the argumentation framework used are presented. Then, an example argumentation dialogue in this domain is provided.

2.1 bCloud Architecture

bCloud is a platform based on the cloud computing paradigm. This platform allows to offer services at the PaaS (Platform as a Service) and SaaS (Software as a Service) levels. On the one hand, the SaaS services are offered to the final users in terms of web applications. On the other hand, PaaS services are offered as web services. The IaaS (Infrastructure as a Service) layer is composed by an physical environment which allows the abstraction of resources shaped as virtual machines. 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.

The system has a layered structure that covers the main parts of cloud computing. Both PaaS and SaaS layers are deployed using an IaaS layer, which provides a virtual hosting service with automatic scaling and functions for balancing workload. The SaaS layer is composed of the management applications for the environment (control of users, installed applications, etc.), and other more general third party applications that use the services from the PaaS layer. At this level, each user has a personalized virtual desktop from which he has access to his applications in the cloud environment, and to a personally configurable area as well. The PaaS layer provides services through REST web services 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.

SaaS Layer. This layer hosts a wide set of cloud applications. All of them make use of the services provides by the PaaS lower layer. bCloud as environment offers a set of native applications to manage the complete cloud environment: virtual desktop, user control panel, and administration panel.

PaaS Layer. The PaaS layer is oriented to offer services to the upper layer, and it is supported by the lower IaaS layer. The components of this layer are: the *IdentityManager*, which is the module of bCloud 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; 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 not to offer infrastructure service as usual cloud platforms do. However, it has to offer this infrastructure service to upper layers of bCloud (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

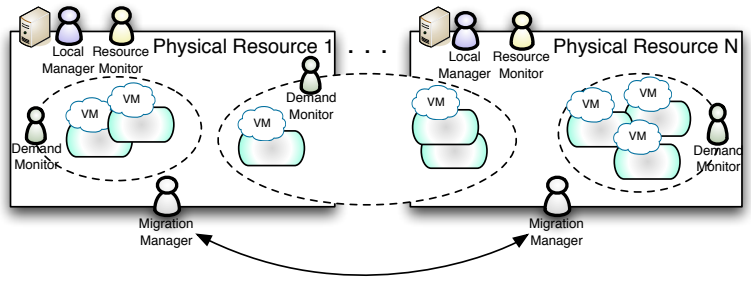

Fig. 1. Virtual Organizations for Elastic Management of Resources

unlimited, but they are supported by a limited number of physical resources, although the final user has the view of unlimited resources.

bCloud MAS. In bCloud, the elastic management of the available resources has been done by a MAS based on Virtual Organizations (VO). In the bCloud 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 1 presents the different roles that bCloud agents can play. These are the following:

- Demand Monitor: in charge of monitoring each demand service which is offered by bCloud (FSS, OSS, Applications of the SaaS layer, etc.). There is one agent of this type per each kind of service. This agents will be able to offer information about not only the instant demand, but also the historical of the demand.
- Resource Monitor: in charge of knowing the usage level of the virtual resources of each virtual machine. There is one in each physical server.
- Local Manager: in charge of allocating the resources of a single physical machine among its virtual machines. There is one in each physical server.
- Migration Manager: in charge of negotiating with their peers the sharing of global resources in terms of 1) Migration of virtual machines between physical servers; and 2) Creation/Destruction of physical resources (switch on and off physical servers), as well as virtual resources (instantiating/stopping virtual machines). There is one in each physical server.

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. In this situation, several potential agreement processes can be identified, let us call them intra-machine or inter-machine. On the one hand, at the intra-machine level, the local manager could start an agreement process with the migration manager to decide the best option among re-distributing physical resources between existing virtual machines, instantiating new virtual machines

or switching on new physical resources (extra physical machines). On the other hand, at the inter-machine level, if the outcome of the former process entails the migration of a virtual machine or the switching on of a new physical server, several migration managers could start an agreement process to decide which machine(s) (already running or not) should host the migrated virtual machine and with which distribution of resources. Summarizing, we can identify several types of solutions as potential outcomes of the agreement processes established:

- *Basic Solution*: consists of redistributing the physical resources of the machine among its virtual machines
- *Halfway Solution*: consist of instantiating a new virtual machine in the same physical machine.
- *Complex Solution*: consist of migrating a virtual machine to a new physical machine that can host it (already running or not).

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 work 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 maximise its success.

2.2 Argumentation Framework

In [1, Chapter 3] we presented a computational case-based argumentation framework that agents can use to manage argumentation processes. In this work, we apply this framework to model the argumentation dialogue among agents in the bCloud platform. Concretely, let us propose a running example where several migration managers try to reach an agreement about the best solution to apply for the resource re-distribution problem during a service peak of demand. The *solution* that an agent proposes defines its *position*. We assume that a problem can be characterised by a set of features that describe it. Also, in our example domain, we assume that all agents are collaborative and all 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. Thus, the *proponent* engages in a two-party argumentation process with the *opponent*, trying to justify its reasons to believe that its solution is most suitable. This section summarises the main components of the framework and illustrates them in this example. The full explanation about the reasoning process that agents use to manage their positions and arguments is

out of the scope of this paper and can be found at [6]. Our case-based argumentation framework defines two types of knowledge resources that the agents can use to generate, select and evaluate arguments:

A case-base with domain-cases: that represent previous problems and their solutions. Agents can use this knowledge resource to generate their positions in a dialogue and arguments to support them as 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. In our example, each migration manager has an individual domain-cases case-base that can query to generate its positions.

A case-base with argument-cases: that store previous argumentation experiences and their final outcome. Argument-cases have three main objectives: they can be used by agents 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. In our example, each migration manager has an individual argument-cases case-base that can query to select the most suitable argument in each step of the process.

Arguments. In our proposal, arguments that agents interchange 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* [7], 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. For instance, values in our example could be considered as types of solutions. Then, an agent could prefer to promote *"ComplexSolutions"* over *"BasicSolutions"*, since it knows by experience that the former type of solutions achieve more successful results in re-distributing resources for an overload service.

The support set S can consist of different elements, depending on the argument purpose. On one hand, 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 and 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* and *argument-cases*). 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* or *counter-examples*, as proposed in [7]. A distinguishing premise is 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

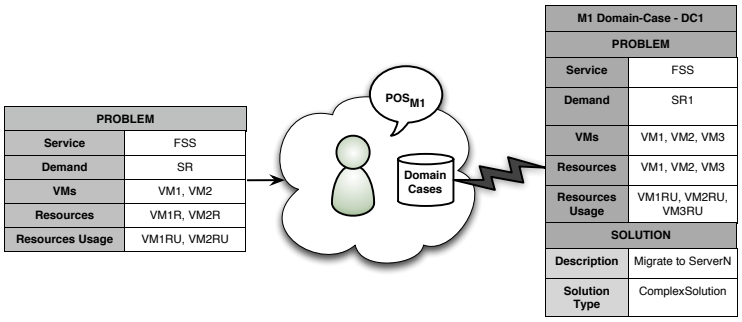

Fig. 2. Example Structure of a Domain-Case

(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. This other type of argument is called an *attack argument*.

Domain-cases. The structure of domain-cases that an argumentation system that implements our framework has depends on the application domain. As example, Figure 2 presents a potential domain-case that the migration manager *M1* could retrieve to generate its recommended solution *"Migrate to ServerN"* (since it has deemed it as similar enough to the current problem), which proposes to migrate the most overloaded virtual machine to another server with more available resources and promotes *"ComplexSolution"* type of solution. Note that in this example we assume that domain-cases also store the type of solution that they represent. The premise *"VM3"* would be a distinguishing premise between the domain-case shown in Figure 2 and another domain-case, say *DC2*, that has exactly the same problem description than *DC1*, but that stores different data for this premise (a different set of virtual machines that provide the service). Also, let us assume that the domain-case *DC2*, which has the same problem description than *DC1*, proposes an alternative solution (e.g *"Instantiate a new VM"*) that promotes the type of solution *"HalfWaySolution"*. Therefore, *DC2* could be used to generate a counter-example attack to *DC1* and vice-versa.

Argument-cases. Argument-cases store the information about a previous argument that an agent posed in certain step of a dialogue with other agents. Their structure is generic and domain-independent. To illustrate the components of an argument-case, let us show the following example. The migration manager *M1* has generated a support argument *SA1* to persuade another migration manager *M2* to accept his position. The position of *M1* (POS_{M1}) was generated by using the domain-case *DC1* and consist of a migration of the most overloaded virtual machine to another physical sever, which promotes the same type of solution than *DC* promotes, say *"ComplexSolution"*. If $p_1 = \{"Service" = FSS\}$, $p_2 = \{"Demand" = SR1\}$, $p_3 = \{"VMs" = \{VM1, VM2, VM3\}\}$, $p_4 =$

PROBLEM	Domain Context	Premises = {p_1, ..., p_5}	
	Social Context	Proponent	ID = M1
			Role = Migration Manager
			ValPref = B<HW<C
		Opponent	ID = M2
			Role = Migration Manager
			ValPref = C<B<HW
		Group	ID = G
			Role = DPC
			ValPref = -
		Dependency Relation = Charity	
SOLUTION	Conclusion = "Migrate to ServerN"		
	Value = C		
	Acceptability Status = Accepted		
	Received Attacks	Distinguishing Premises = ∅	
		Counter Examples = DC2	
JUSTIFICATION	Cases = DC1		
	DG1		

Table 1. Argument-case representing SA1

{"*Resources*" = {$VM1R, VM2R, VM3R$}}, and p_5 = {"*ResourcesUsage*" = {$VM1RU, VM2RU, VM3RU$}} then $SA1$ = {"*MigratetoServerN*", C, < {p_1, p_2, p_3, p_4, p_5}, {$DC1$}, -, -, -, -, - >}.

Table 1 presents an example of an argument-case that stores the information of the support argument $SA1$ in tour running example. In the table, *"B"* stands for *"BasicSolution"*, *"HW"* for *"HalfWaySolution"* and *"C"* for *"ComplexSolution"*. Argument-cases have the three possible types of components that usual cases of CBR systems have: the description of the state of the world when the case was stored (*Problem*); the solution of the case (*Conclusion*); and the explanation of the process that gave rise to this conclusion (*Justification*).

The *problem* description 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 also be kept. The social context of the argument-case includes information about the *proponent* and the *opponent* of the argument and about their *group*. Although we have this distinction between *proponent* and *opponent* of an argument (and of its underlying defended or attacked position), in our running example all agents are willing to collaborate and share the common objective to propose the best solution for the overloaded service problem.

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 this work, we consider two types of dependency relations (inherited from [8]): *Power*, when an agent has to accept a request from other agent because of some pre-defined domination relationship between them (a hierarchy defined over roles, for instace); and *Charity*, when an agent is willing to answer a request from other agent without being obliged to do so. For instance, in the argument-case of Table 1 proponent and opponent play the same role (migration managers) and hence, we have set a charity dependency relation between them. Also, both managers belong to the same group *DPC*, which represents a *"Data Processing Center"* where the bCloud platform is installed and does not impose any specific value preference order over its members.

In the *solution* part, 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 if the argument was deemed *acceptable*, *unacceptable* or *undecided* in view of the other arguments that were put forward in the agreement process. Thus, the conclusion of the argument-case in Table 1 presents the solution proposed by *M1* to solve the service overload problem. In addition, the conclusion part includes information about the possible *attacks* that the argument received during the process. These attacks could represent the justification for an argument to be deemed unacceptable or else reinforce the persuasive power of an argument that, despite being attacked, was finally accepted.

In the example shown in Table 1 we can see that the opponent, say migration manager *M2*, attacked the argument represented by this argument-case with the counter-example *DC2*. Therefore, *M2* generated an attack argument *AA2* with this counter-example in the support set: $AA2 = \{$*"Instantiate a new VM"*, HW, $< \{p_1, p_2, p_3, p_4, p_5\}$, -, -, -, -, -, $\{DC2\} >\}$. However, as shown in Table 1, *M1*'s support argument remained acceptable at the end of the dialogue (when the argument-case that represents it was retained in the argument-cases case-base). 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 and still remained acceptable at the end of the agreement process (see [6] for a detailed explanation of this reasoning process).

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 domain-cases and argument-cases). In the example of Table 1, the justification part of the argument-case includes the domain-case *DC1*, that the migration manager *M1* used to generate its position. In addition, the justification of each argument-case has a *dialogue-graph* (or several) associated, which represents the dialogue where the argument was put forward (*DG1* in Table 1). 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.

2.3 Argumentation Dialogue

The agents of the our framework use an argumentation protocol to manage positions and arguments and perform the argumentation dialogue. The entire protocol is defined in [1, Chapter 4]. Here a simplified version of the protocol proposed in [9] is instantiated to provide an example dialogue between a set of migration agents presented with the problem of deciding the best solution to re-distribute resources for the overloaded service. The protocol is represented by

a set of locutions that the agents use to argue with other agents, and an state machine that defines the allowed locutions that an agent can put forward in each step of the argumentation dialogue (presented in Figure 3). 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, ϕ) (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, ϕ) (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 show how our argumentation framework can be applied to manage the service overload problem, the data-flow for the argumentation dialogue among several migration agents engaged in the agreement process is described below. The process starts when a demand monitor agent notifies an overload in a service that it manages. Then, the resource monitor agent in charge of the virtual machines that offer this service sends the load information about the resources associated to these virtual machines to the migration manager of the physical machine that hosts them. For clarity reasons, we assume in this example that all virtual machines that offer a service are hosted in the same physical machine. However, in a real implementation, these virtual machines could be distributed in several physical machines and in that case, several parallel argumentation dialogues would be started by the migration manager of each physical machine. Also, migration managers are connected among them and are able to check the positions proposed by other migration managers.

1. The first state is the initial state. At the beginning of the agreement process, migration managers remain in this state waiting for an *open_dialogue* locution. Also, they will come back to this state when the agent that starts the argumentation dialogue (say, the *initiator* agent) communicates that the dialogue has finished. The *open_dialogue* locution informs the migration manager agent that receives it about the start of a new dialogue to

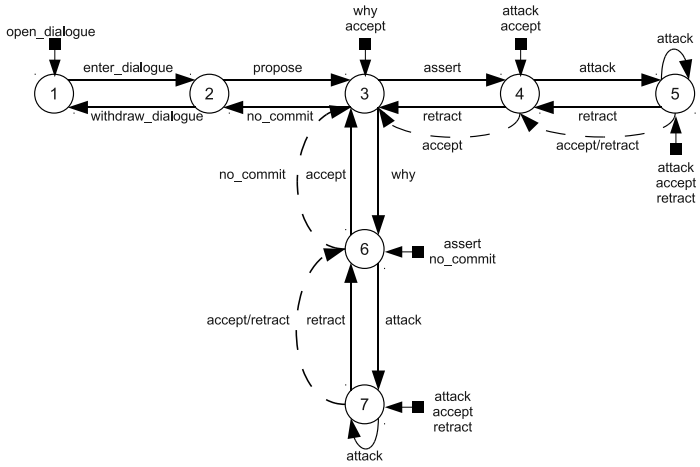

Fig. 3. Argumentation state machine of the agents

solve a re-distribution problem. Then, this agent will retrieve the cases of its domain-cases case-base which features match the given problem characterisation (e.g. the FSS overload presented in Figure 2) with a similarity degree greater than a given threshold. Finally, if the agent has been able to retrieve similar domain-cases and use their solutions to propose a solution for the current problem, it will engage in the argumentation dialogue with the locution *enter_dialogue* and will go to the state 2. A migration manager only engages in the dialogue if it has solutions to propose. For instance, on our running example, *M1* will engage in the dialogue, since it has been able to retrieve *DC1* as a similar case.

2. When a migration manager is in this state it has retrieved a list of similar domain-cases to the current problem to propose a solution (position to defend). If the agent has been able to generate several alternative solutions, it will select the most suitable for its current context (following the reasoning process presented in [6]) and go to state 3 proposing this solution. Otherwise, the agent will use the *withdraw_dialogue* locution and will return to state 1. In the example, *M1* would propose the solution reused from *DC1*, *"Migrate to ServerN"*.
3. In this central state, the migration manager can try to attack other positions or defend its position from the attacks of other agents. First, the agent checks if there is any *why* request from other opponent agent that asks for justifying its proposed solution. The agent that received the *why* locution will *assert* a support argument to its opponent if it is able to do so and goes to state 4 in this case. In our example, *M1* will assert *SA1* when *M2* ask it for a justification of its proposed solution. Otherwise, if the migration manager

is not able to provide a support argument to defend its position it will go to state 2 and try to propose another solution from its list of generated positions. Alternatively, if the migration manager has not received any *why* request, it will ask other migration manager (chosen randomly) that has proposed a different position for an argument to justify it, using the *why* locution. In that case, the agent will pass to state 6. Again, in our example it would occur if *M1* has not received any justification requests and asks *M2* to justify its solution.

4. In this state, the migration manager that has put forward a support argument for its position waits for an *attack* or an *accept* locution. After certain time has passed and no locution is received, the agent will return to state 3. In the case that an attack is received, the migration manager will try to generate another attack to rebut the attack argument received. If the migration manager is not able to counter-attack, it will retract its support argument and go to state 3. Otherwise, it will go to state 5 to wait for another attack or a retraction from the opponent. In our example, as presented in Table 1, *M1* receives an attack from *M2* that consists of the counter-example case *DC2*, which proposes a different solution for the same problem characterisation.
5. In this state a proponent and an opponent migration managers are engaged in an attack phase arguing about the position of the proponent. When possible, every attack received will be replied with another attack and the proponent agent will remain in this state. However, when the proponent cannot reply an attack argument with another attack, it will have to retract its last attack and go to state 4. Otherwise, if the proponent receives an *accept* locution, the opponent migration manager accepts its last given attack. That implies going back to state 4, where the opponent agent must accept the proponent support argument and its position. As shown in Table 1, the justification argument *SA1* was accepted at the end of the dialogue, although being received an attack with the counter-example *DC2*. This means that *M1* was able to rebut the attack and *M2* wasn't able to counter-attack. For instance, if we assume that $VMs = \{VM1\}$ in *DC2*, it could happen if after the attack of *M2*, *M1* counter-attacks with a distinguishing premise on the *VMs* attribute (since it does not subsume the exact value of this premise in the problem reported, while this premise in the characterisation of *DC1* does).
6. When a migration manager enters to this state it waits for an *assert* or a *no_commit* locution. Here, after a specific time has passed and no locution is received, the agent will return to state 3. Alternatively, if the proponent migration manager receives an *assert* locution from an opponent and it is not able to *attack* the support argument received with this locution, it will accept the opponent position and go to state 3. However, if an attack argument can be generated, it sends it to the opponent and goes to state 7. Otherwise, if a *no_commit* locution is received in this state the opponent agent retracts its position and the proponent goes back to state 3. In our running example, *M1* will wait in this state for a justification argument from *M2*, which will include *DC2* as supporting element and, if provided, will attack this argument with the counter-example *DC1* or the distinguishing premise *VMs*.

7. In this state, the migration manager that has sent an *attack* argument to an opponent migration manager waits for a counter-attack or an *accept* locution from it. The proponent will try to reply to any attack received for its attacking arguments and remain in this state while it can generate new attacks. If an *accept* locution is received the last attack argument of the proponent migration manager has been accepted by the opponent, thus the opponent support argument is defeated and the proponent will go to state 6 to wait for another support argument from the opponent or a *no_commit* locution. Nevertheless, if an attack of the opponent cannot be replied, the agent has to accept the opponent attack argument, retract its attack argument and go to state 6. Then, the proponent must go to state 3 after accepting the opponent position. In the example, after attacking *M2*, *M1* will wait in this state for a counter-attack. If we assume that *M2* is not able to generate another attack nor more justification arguments, *M2* would retract its attack argument and withdraw its position from the dialogue.

The dialogue finishes when no new positions or arguments are proposed after a certain time. Then, the initiator migration manager 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 draw, the final solution will be the most frequent position generated by the migration managers during the argumentation dialogue. Finally, once a position is selected as the outcome of the agreement process, the migration manager 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 Discussion

Cloud computing is a new model of business and technology services, which allows the user to access a catalog of standardized services and meet the needs of his business in a flexible and adaptive way, paying only for the consumption made [10]. Multi-agent systems are a technology paradigm where a set of intelligent software agents interact to cope with complex problems in a distributed fashion [11]. In the literature, there are not many references of work that combines both agents and cloud computing paradigms: in [12] software agents figure as a new cloud computing service which would represent clients in virtual environments; [13] presents a complex cloud negotiation mechanism that supports negotiation activities in interrelated markets; [14] proposes the integration of a cloud on GRID architecture with a mobile agent platform that is able to dynamically add and configure services on the virtual clusters; and [15] presents a service-oriented QOS-assured cloud computing architecture. These contributions pave the way for an interesting new area of investigation on cloud-based multi-agent systems. Recent research foresees the advent of a new discipline of

agent-based cloud computing systems on the Future Internet. This work identifies research opportunities from the Multi-Agent Systems (MAS) technology to the cloud computing paradigm and vice-versa. Summarising, agents can provide clouds with intelligent, flexible, autonomous and proactive services and clouds can provide agents with processing and memory resources at unprecedented scale [2]. Argumentation-based agreement technologies, as a proficient research area within MAS-based agreement models, should echo these opportunities and contribute to the achievement of new challenges in agent-based cloud computing. On the next generation of agreement technologies we envisage systems of humans and agents with the ability of reaching agreements by using the unlimited computational resources provided by cloud systems. This entails the development of new algorithms, tools and models that enable the creation of open systems where virtual agents and humans coexist and interact transparently into a fully integrated agreement environment. Research on this area will provide answers to questions like: What is necessary to know and design for software agents and humans to interact in an agent-based cloud computing systems? and how these interactions should be formalised and structured to obtain software products that are effective in such environments? Argumentation-based agreement technologies can contribute to cope with these open issues.

In the last years, the community of argumentation in MAS has advanced research in many fields on the area of applying argumentation theory to harmonise agents incoherent beliefs and model the interaction among a set of agents [16], but the application of argumentation approaches to cloud computing is a new challenge. Research on argumentation in Grid computing[3] can be viewed as related work, in the sense that grid computing and cloud computing have many similarities and a main difference in the storage conception: the grid is well suited for data-intensive storage while in cloud systems the amount of data stored can be large or not, depending on the user needs. Thus, results from this research can be studied and analysed as an starting point to foster work in argumentation-based agreement technologies applied to cloud computing. In this paper, we propose the application of argumentation as a suitable technology to model load-balancing services in cloud systems. Concretely, 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. 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]). Current work is being performed to implement and test this system, in order to analyse the viability and advantages of this approach. Also, the advantages that this approach contributes over direct resource allocation algorithms must be analysed. However, our intuition is that both direct resource allocation algorithms and argumentation-based techniques could be considered as complementary. In our scenario, migration managers could use different decision making policies (experience-based or any other kind of resource allocation techniques) to propose the best solution to the resource re-distribution problem

[3] `http://www.argugrid.org/`

and then, engage in an argumentation dialogue to decide which is the best solution to apply. Further work 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 be: negotiating Service Level Agreements; providing a method to harmonise 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.

Acknowledgements This work is supported by the Spanish government grants [CONSOLIDER-INGENIO 2010 CSD2007-00022, TIN2008-04446, and TIN2009-13839-C03-01] and by the GVA project [PROMETEO 2008/051].

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. Barham, P., et. al.: Xen and the art of virtualization. In: 9th ACM Symposium on Operating Systems Principles, ACM Press (2003) 164–177
5. 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
6. Heras, S., Jordán, J., Botti, V., Julián, V.: Argue to agree: a case-based argumentation approach. International Journal of Approximate Reasoning (In Press)
7. Bench-Capon, T., Sartor, G.: A model of legal reasoning with cases incorporating theories and values. Artificial Intelligence **150**(1-2) (2003) 97–143
8. Dignum, F., Weigand, H.: Communication and Deontic Logic. In: Information Systems Correctness and Reusability, World Scientific Pub. Co. (1995) 242–260
9. Jordán, J., et. al.: A customer support application using argumentation in multi-agent systems. In: 14th Int. Conf. on Information Fusion. (2011) 772–778
10. : The future of cloud computing. Technical report, European Comission (2010)
11. Wooldridge, M., Jennings, N.R.: Intelligent agents: Theory and practice. The Knowledge Engineering Review **10**(2) (1995) 115–152
12. 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
13. 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
14. 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
15. 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
16. Rahwan, I., Simari, G., eds.: Argumentation in Artificial Intelligence. Springer (2009)