Real-Time Agreement and Fulfilment of SLAs in Cloud Computing Environments

Fernando de la Prieta [a,*], Stella Heras [b] and Javier Palanca [b] and Sara Rodríguez [a] and Javier Bajo [c] and Vicente Julián [b]

[a] *Department of Computer Science, University of Salamanca, Plaza de la Merced s/n, 37008, Salamanca (Spain)*
[b] *Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia (Spain)*
[c] *Departamento de Inteligencia Artificial, Facultad de Informática, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Madrid, (Spain)*

A Cloud Computing system must readjust its resources by taking into account the demand for its services. This raises the need for designing protocols that provide the individual components of the Cloud architecture with the ability to self-adapt and to reach agreements in order to deal with changes in the services demand. Furthermore, if the Cloud provider has signed a Service Level Agreement (SLA) with the clients of the services that it offers, the appropriate agreement mechanism has to ensure the provision of the service contracted within a specified time. This paper introduces real-time mechanisms for the agreement and fulfilment of SLAs in Cloud Computing environments. On the one hand, it presents a negotiation protocol inspired by the standard WS-Agreement used in web services to manage the interactions between the client and the Cloud provider to agree the terms of the SLA of a service. On the other hand, it proposes the application of a real-time argumentation framework for redistributing resources and ensuring the fulfilment of these SLAs during peaks in the service demand.

Keywords: Cloud Computing, Service Level Agreements, Multi-Agent Systems, Virtual Organisations, 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 the technological level, many difficulties have been overcome with the use of virtualisation of hardware resources [6]. However, how to assign the physical infrastructure among virtual machines is a current topic in some research fields [47]. This raises the need for designing protocols that provide the individual components of the Cloud architecture with the ability to self-adapt and to reach agreements in order to deal with changes in the demand of services. Furthermore, if the Cloud provider has signed a Service Level Agreement (SLA) with the clients of the services offered in the platform, the appropriate agreement mechanism has the added difficulty of ensuring the provision of the service contracted within a specified time. Therefore, in these environments there is a need for coordinating the entities that make use of the available resources in the Cloud and reaching agreements in a *real-time* fashion.

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 [42]. The possibility of applying Multi-Agent Systems (MAS), based on Virtual Organisation (VO) has many advantages due to the ability of these systems to adapt themselves in the presence of incomplete information in 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 real-time argumentation processes to collaboratively solve problems [30]. The use of argumentation techniques in the context of Cloud Computing raises interesting challenges. One is the hard constraints put on the efficiency of the process. Another one is the fact that arguments provide useful justifications to the potential clients when agreements are obtained. Therefore, 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 real-time agreement mechanisms for the agreement and fulfilment of SLAs in Cloud Computing environments. On the one hand, it presents a negotiation protocol inspired by the standard WS-Agreement used in web services to manage the interactions between the client and the Cloud provider to agree the terms of the SLA of a service. On the other hand, it proposes the application of a real-time argumentation framework for redistributing resources and ensuring the fulfilment of these SLAs during peaks in the service demand.

At the external level, a Cloud Computing system is composed of a set of Software, Platform and/or Infrastructure services which are offered to the users, commonly known as XaaS (X as a Service) [38] (see Figure 1). The software and platform services can be considered as web applications that are deployed over the Cloud platform and store their persistent data by using the services provided by the Cloud. In this study, we use the term *service* to refer to each of these applications. Infrastructure services, on the other hand, are a way of offering computational resources on the Cloud. At the internal level, the system consists of a set of physical machines (servers). These machines can be available (or turned on) or unavailable (or turned off). The available physical machines host abstractions of hardware components called virtual machines. Both physical and virtual machines are connected among them through an internal network. The features and topology of this network are out of the scope of this work.

By definition, a Cloud Computing environment is a high availability system, which means that the quality and availability of its services must be ensured independently of their demand. This means that each of the services at the PaaS or SaaS level have to be deployed in n virtual machines that are distributed among m physical machines ($m \geq 2$). In this way, the consumption of virtual resources (and their associated physical ones) varies depending on the demand and hence, the number of available virtual machines and physical servers hosting them must be increased or decreased accordingly; which is referred to as elasticity of the Cloud system.

The latest generation of virtualisation environments allows the resources available in the physical machines to be dynamically allocated among the virtual machines according to the current needs [6]. Nowadays, the greatest challenge in a Cloud environment is how

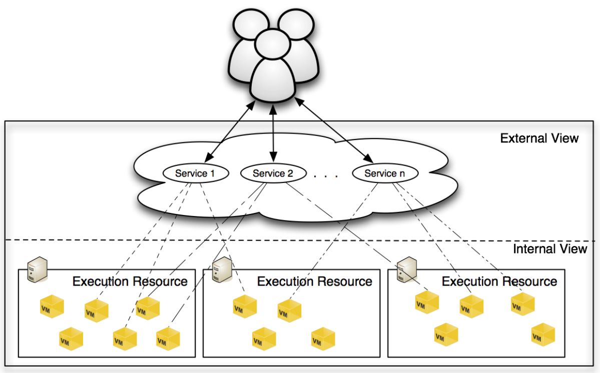

Fig. 1. External and internal view in a Cloud Computing environment

to efficiently redistribute 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. This means, to switch on or off virtual machines in a specific physical server at any given moment, or even, to migrate virtual machines in execution between physical servers. Furthermore, Cloud providers can offer Software or Platform services that are restricted to a maximum execution time, that is, the service must be provided before a deadline is met. Commonly, companies sign SLAs with their customers. Among other terms, these SLAs specify a maximum acceptable time within the customer must be provided with the service contracted in the company. Therefore, if the Cloud provider has signed a SLA with its Cloud customer, it must ensure that the service(s) contracted will be provided within a specific time. For instance, this can be the case of customer support companies that have contracted their storage or computing services with the Cloud provider.

In the next sections we present our Cloud Computing platform that is able to meet SLAs with its users: Section 2 presents +Cloud, a new Cloud Computing platform based on MAS; Section 3 explains the real-time agreement processes performed by the agents of this platform to deal with the problem of resource redistribution during a peak service demand; Section 4 evaluates our proposal with different tests; finally Section 5 introduces related work and future challenges and summarises the results of this study. Table 4 shows the list of acronyms used throughout this paper.

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 internal layer, which provides a virtual hosting service with automatic scaling and functions for balancing workload. Therefore, this platform does not offer a IaaS (Infrastructure as a Service) layer. The internal layer is formed by the physical environment which allows the abstraction of resources shaped as virtual machines. Thus, since this layer is internal to the system, +Cloud does not offer this kind of service to its 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 personalised 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 [33] 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 hierarchy. 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;
∗ and the *Object Storage Service (OSS)*, which provides a simple and flexible schemaless database service oriented towards documents.

The virtual and physical resources are managed dynamically. To this end, a virtual organisation of intelligent agents that monitor and manage the platform resources is used. This organisation implements an argumentation-based agreement technology in order to achieve the distribution of resources depending on the needs of the whole 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 performs the redistribution of resources among virtual machines hosted within a single host. That is, a physical server has a set of available physical resources (processor, memory and disk) that have to be shared between different virtual machines hosted in this server, leaving a minimal set of resources available to the host itself. At this level, the main objective is to maximise the use of resources and hence, to release those that are not really needed. At the macro level, however, the system performs the global redistribution of resources within the Cloud. This means that starts or stops virtual machines on physical servers, or even migrates machines between physical servers. At this macro level, the goal is to minimise the use of resources, so that the largest number of machines remain disabled, but always keeping the high availability of the system 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.

In +Cloud, the elastic management of the available resources is performed by a MAS based on VO, which eases the design of the overall system [20]. 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. In addition, one of the most innovative aspects of the +Cloud framework is the fact that it provides the possibility of reaching agreements at two different levels: at the *external level*, +Cloud allows temporal SLA's parameter negotiation between the system and its customers using multi-agent interaction protocols; at the *internal level*, +Cloud includes a computational case-based argumentation framework that agents can use to engage in an agreement process to make a decision about the best resource redistribution to meet a SLA in a peak service demand.

Figure 2 presents the different roles that +Cloud agents can play. These are the following:

– *Global Regulator (GR)*, in charge of agreeing the redistribution of resources at a macro level within its peers. This includes 1) the migration of virtual machines between physical servers; 2) the creation/destruction of physical resources (turn

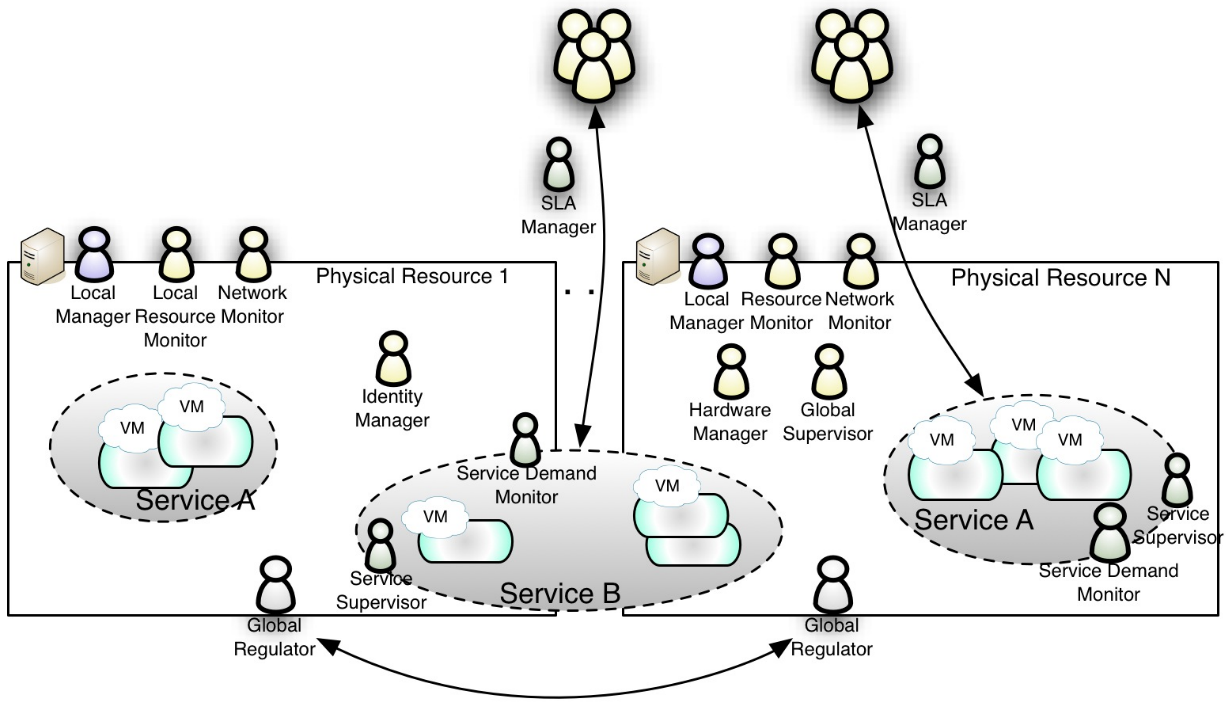

Fig. 2. Virtual Organisations for Elastic Management of Cloud Resources

on/turn off physical servers), as well as virtual resources (instantiating/stopping virtual machines); or simply (3) the redistribution of the resources in each physical server among its virtual resources. The main objective of the agreement process is to find a solution to deal with problems on the demand of the services, respecting the SLAs. The minimum amount of physical servers should be used, so that less energy is consumed. There is one agent playing this role in each physical server.

– *Global Supervisor (GS)*, in charge of supervising and ensuring that the other agents of the VO work correctly. If an error occurs, or if one of the agents does not respond to its incoming messages, this role should take the necessary actions to restore the system to a stable state. Also, it may act as a judge if GRs cannot agree on the redistribution of a particular service. There is one agent playing this role for each VO of +Cloud.
– *Hardware Manager (HM)*, in charge of managing 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. There is one agent playing this role for each VO of +Cloud.
– *Identity Manager (IM)*, in charge of supervising the module of +Cloud that offers authentication services to clients and applications. There is one agent playing this role for each VO of +Cloud.
– *Local Resource Monitor (LRM)*, in charge of knowing the usage level of the virtual resources for each virtual machine. There is one agent playing this role in each physical server. This agent 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 agent playing this role in each physical server. The knowledge that this agent needs is provided by the LRM of the same physical machine. It is able to redistribute the resources among the virtual machines based on its own

knowledge and can also turn on or off the virtual machine of a given service.

– *Network Monitor (NM)*, in charge of monitoring the network from the point of view of each single physical machine. This information allows the system to make better redistribution decisions in an argumentation dialogue, in view of the network load. There is one agent playing this role in each physical server.
– *Service Demand Monitor (SDM)*, in charge of monitoring each service demand that is offered by +Cloud, which means a service at the SaaS and PaaS levels. There is one agent playing this role per each kind of service. These agents are able to offer information about the current and past demands. Also, they incorporate a load balancer to redirect the request to the different virtual machines which are offering a service at a specific time.
– *Service Supervisor (SS)*, responsible for making decisions about each individual service. These agents receive information from the SDM of the service that they supervise, and must perform actions to ensure quality levels to fulfil the SLA contracted, to fix possible errors, or to ensure the high availability of the service. In +Cloud, this high availability entails to have at least two virtual machines per service, located in two different physical machines. There is one agent playing this role for each service and it is located at the same virtual machine that hosts the SDM of the same service.
– *SLA Manager*, in charge of offering a set of communication facilitation services to other client agents (representing the customers of the Cloud system) using some knowledge about the requirements and capabilities of those agents. A client agent can ask the SLA Manager to reach an agreement about the offered services. The interaction between the client agent and the SLA Manager is divided into two stages: negotiation and execution (see Section 3.1). There is one agent playing this role for each service offered by the Cloud platform.

3. RT Agreements in +Cloud

As pointed out before, the agents of the +Cloud platform engage in agreement processes to provide services. At the external level, customers negotiate with the SLA manager the terms of the service contracted. During the execution of the service, a peak in the demand can give rise to an overload of one or more of the virtual machines that provide this service. Therefore, the system has to redistribute its virtual and physical resources to cope with this problem and be able to fulfil the maximum time contracted in the SLA to provide the service. As indicated, resource redistribution can be done at the micro level (intra-machine) or at the macro level (inter-machine). Therefore, at the internal level, global regulators engage in argumentation processes to decide the best redistribution to solve the overload of a service. This section presents the operation of +Cloud at both levels.

3.1. *External RT Agreements: SLAs Negotiation*

This section presents the needed interactions between the client and the Cloud provider to agree the terms of the SLA of a service. These interactions follow a negotiation protocol between an agent representing the client and the SLA manager. The negotiation protocol provides more flexibility in order to adapt the value of the service execution terms in the initial proposal to a more suitable value for the client and the service provider.

As explained in Section 2, a client agent can ask the SLA Manager to reach an agreement on the quality of the offered services. The interaction between the client agent and the SLA Manager is divided into two stages: *negotiation* and *execution*. In the first stage the client and the SLA Manager negotiate the terms of the execution of the service that is going to fulfil the client's request. Once the client agrees, the second stage starts. In this stage the Service Supervisor executes the service, taking into account the agreement established by the SLA Manager and a valid identification provided by the Identity Manager. Below, both stages are explained with more detail.

3.1.1. *Negotiation*

The interaction between the client and the SLA Manager is made through a negotiation protocol inspired by the standard WS-Agreement used in web services [1]. To do that, the SLA Manager has a repository with the service templates provided by the Cloud Providers. These templates contain descriptions with the information related with functional service parameters (e.g. Inputs, Outputs, Preconditions, and Effects) and non-functional service parameters (e.g. service execution time, which represents the SLA requested by

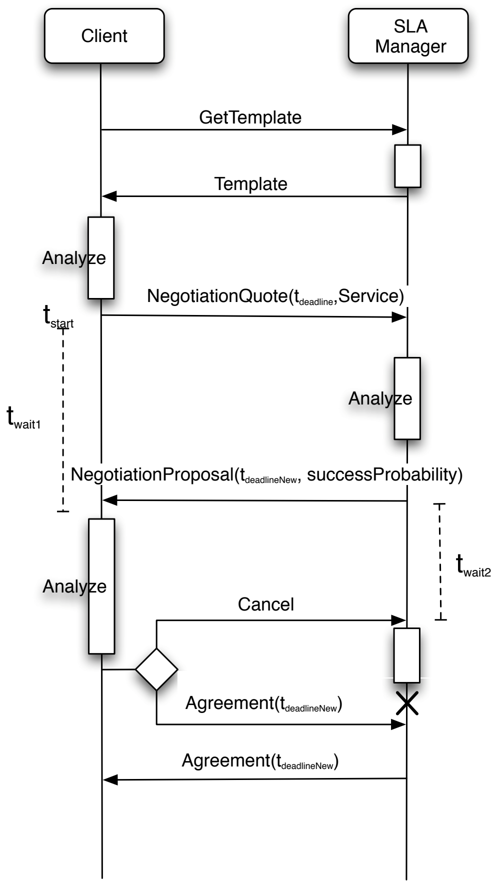

Fig. 3. Negotiation Protocol

the client). In order to offer this information to the client, an OWL-S service description extended with the non-functional parameter is used. This description includes the service execution time. The value of this parameter is represented by an average execution time. For new services, this execution time is obtained by measuring off-line the cost of the service. After that, the SLA Manager makes online estimations of the actual execution time of the services provided and the expected execution time of each one is computed as the average time of these estimations. The interactions of the negotiation protocol between the client and SLA Manager are shown in Figure Figure 3:

– The SLA Manager publishes the templates of the services provided by the +Cloud platform.
– A client queries the agreement templates.
– Based on a suitable template, the client analyses the service execution time t and, considering the value of this term, creates a new proposal providing the desired deadline ($t_deadline$) in absolute time (date and hour) before which the service should finish its execution. The offer is sent to the SLA Manager. The query includes a template formatted as an OWL document, which contains a set of service inputs, outputs and the deadline:

  $query = \langle \{input\}, \{output\}, deadline \rangle$

  The client waits for an answer until a timeout (t_wait1) is reached. If the client does not receive an answer, it cancels the negotiation.
– The SLA Manager analyzes the client's template. This process is temporal bounded. Basically, the SLA Manager checks with the Service Supervisors if the service requested by the client is available and if it can be provided before the client deadline. Each service consists on a set of elements:

  $service = \langle serviceId, \{input\}, \{output\}, serviceDuration, probability \rangle$

  If the service can be provided on time, the SLA Manager returns the same deadline. Otherwise, it proposes a new deadline ($t_deadlineNew$). In both cases, the SLA Manager also returns the probability of service execution success before the deadline. The success probability of a service is calculated using the success probability of each action that must be carried out in the service execution. The SLA Manager selects the service, taking into account the success probability of the service and its execution time. In order to guarantee that required service is going to be provided on that deadline, the SLA Manager schedules its service execution. Then, it waits for an answer from the client during a time ($_twait2$). If the SLA Manager does not receive an answer, it removes the scheduled service execution.
– The SLA Manager sends the service template with its deadline to the client. If the client agrees with the proposed plan (its deadline –SLA– and the service probability of success is higher than a client's threshold) the client sends an agree message. Finally, the Service Supervisor starts to accept requests within the agreement established by the SLA Manager. Then, the SLA Manager notifies the Identity Manager that this client is authorised to use the service in the terms defined by the SLA. In the case whereby the client does not agree with the agreement proposed, the client

sends a message to the SLA Manager and it removes the service from the schedule and, if possible, proposes a new one.

3.1.2. Execution

If the client accepts the proposed service parameters, the Service Supervisor begins to accept requests for these services and establishes a formal commitment to force the execution of the service according to the agreed terms. The most important requirement is the value of the SLA, which implies that the service must be served before that deadline. To ensure that the deadline of each agreement is met, the Local Manager agents act as hypervisors of the virtual machines running in the physical resources. These agents schedule the computational resources according to the agreements established by the SLA Manager.

Ensuring these temporal commitments is a complex task since we can not predict which is going to be the future workload demand because the arrival rate of service requests is unknown in an open system. The Local Managers need to predict when a service execution is going to be finished in order to know if the temporal commitment proposed by the SLA Manager is going to be fulfilled. To do this, their scheduling algorithm implements the Deadline Prediction Scheduler [32]. This scheduling algorithm uses resource booking to ensure that a temporal commitment is accomplished. In order to fulfil a temporal commitment, the Local Managers need to guarantee that the task will have enough time in the processor. The processor time is a valuable resource and is directly related to the instant of time when a task will finish its execution. Thus, if we make a booking of the processor resource that could never be diminished when we establish a temporal commitment, we can ensure that the temporal commitment will be fulfilled.

3.2. Internal RT Agreements: Resource Redistribution

In [30], a *real-time argumentation framework* that agents can use to engage in an agreement process to make a decision about a problem at hand within a bounded period of time is presented. In this section, we apply this framework to model the argumentation dialogue among agents in the +Cloud platform at the internal level. In this case, thus, the problem to solve is to decide the best redistribution of resources to cope with an overload in a service, taking into account the SLA contracted with the client.

On the one hand, from the intra-machine perspective, the redistribution of resources is made by the Local Manager agent based on information provided by the Local Resources Monitor agent. Both agents are located in the same physical machine and follow a case-based reasoning approach to perform the resource reallocation. The underlying process to reallocate resources at this level is out of the scope of this paper.

On the other hand, when a Local Manager detects that the physical server has insufficient resources to meet the demand for the service, or when an Service Supervisor notices that the quality of service is not adequate to fulfil the SLA contracted, an inter-machine redistribution of resources is necessary. Furthermore, the Local Manager agent can detect that the physical machine is underused. In that case, it may try to perform a redistribution to maximise the usage of its resources or otherwise, turn off the physical machine to save energy.

To deal with these situations, the Local Manager reports the problem to the Global Regulator of its physical machine, which starts an agreement process with the other Global Regulators of +Cloud in order to decide the best option for redistributing physical resources among existing virtual machines in the whole Cloud platform. Several types of "solutions" can be identified as potential outcomes for the agreement processes established:

– *Basic Solution*, consists of redistributing resources inside the same physical machine.
– *Easy Solution*, consists of instantiating a new virtual machine of a particular service on a physical machine that has sufficient resources to serve this demand.
– *Half Solution*, consists of starting a new physical machine, and instantiating a new virtual machine for a particular service.
– *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 is not overloaded.
– *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 is not overloaded.

Any of these solutions entails an underlying argumentation process between the Global Regulators of the physical machines of the VO to reallocate virtual and physical resources to solve the overload problem. However, in this work we do not aim to discuss the best mechanism to implement the solution agreed (the actual technical operations to put into action such solution), but to provide the agents of +Cloud with the ability of engaging in an *argumentation* dialogue to collaboratively decide which would be the best solution to make *before* starting the process to actually implement it. Our hypothesis is that agents may make the most of their experience and help each other to avoid complex agreement 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 subsequent implementation process and maximise its success.

In the +Cloud platform, Global Regulators implement a real-time argumentation framework that allows them to have an argumentation dialogue with their peers and learn from the experience (saving the knowledge acquired as cases in case-bases). Also, they use a real-time argumentation process to manage their positions and arguments during the argumentation dialogue. By this process, agents exchange ACL messages trying to reach an agreement about the best redistribution of resources to solve the service overload 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 [30]. This section summarises the main components of the real-time argumentation framework and the argumentation process that Global Regulators use to exchange their positions and arguments in +Cloud, illustrating the process with an example.

3.2.1. Argumentation Framework

Our real-time argumentation framework defines three types of individual knowledge resources that the agents can use to manage arguments:

- **A case-base with domain-cases,** that represent previous problems and their solutions. Agents can use this knowledge resource to generate their positions and arguments. The *position* of an agent represents the *solution* that this agent proposes, by reusing the solution applied to solve a similar problem in the past. Also, agents increase their domain knowledge at the end of each real-time argumentation dialogue by adding new cases to their domain-cases case-base.
- **A case-base with argument-cases,** that store previous argumentation experiences and their final outcome. Agents use this resource to select the best position and argument to put forward in a specific situation in view of how suitable a similar position or argument was in a similar real-time argumentation dialogue. Also, agents store the new argumentation knowledge gained in each real-time agreement process, improving the agents' argumentation skills.
- **A set of argumentation-schemes,** that represents stereotyped patterns of reasoning [46]. 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 a 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:

Definition 3.1 (Argument) *$Arg = \langle \phi, v, \{S\} \rangle$, where ϕ is the conclusion of the argument (the solution promoted by 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* [10], 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 redistributing 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 redistributes 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 [16]):

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. *Authorisation*, 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. once the SLA has been signed, the SLA Manager is *authorised* to demand the fulfilment of the SLA contracted of the Service Supervisor); 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).

Definition 3.2 (Support Set) *A support set for an argument consists of a set of elements:*
S = { {*premises*}, {*domainCases*}, {*argumentCases*}, {*argumentationSchemes*}, {*distinguishingPremises*}, {*counterExamples*}, {*criticalQuestions*} }

The support set S can consist of different elements, depending on the argument purpose. 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*. This other type of argument is called an *attack argument*.

The following functions provide a formal definition for the different types of attacks. First, let us assume that we have a problem to solve denoted as P, a set of cases denoted as $C = \{c_1, c_2, ..., c_n\}$, a set of premises denoted as $F_i = \{f_1, f_2, ..., f_m\}$ such that $f_j \in P$ represents a premise that describes the problem P and $f_i \in c_i$ represents a premise that describes the case c_i ($f_i, f_j \in F_i$, thus, we represent both problems and the problem description of cases with a set of premises), a function $value_{c_i}(f_i)$ that returns the value of a premise in a case $c_i \in C$ (i.e. the actual data of that premise, do not confuse with the notion of value promoted by arguments), and a function *conclusion(c_i)* that returns the conclusion of the case c_i (i.e. the solution promoted by the case).

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.

Definition 3.3 (Distinguishing Premise) *A distinguishing premise f_i with respect to a problem P between two cases $c_1, c_2 \in C$ is defined as:*
$\exists f_i \in c_1 \wedge \nexists f_i \in P \mid \exists f_i \in c_2 \wedge value_{c_1}(f_i) \neq value_{c_2}(f_i)$
or else, $\exists f_i \in c_1 \wedge \exists f_i \in P \mid value_{c_1}(f_i) = value_P(f_i) \wedge \nexists f_i \in c_2$

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.

Definition 3.4 (Counter-Example) *A counter-example for a case $c_1 \in C$ with respect to a problem P is another case $c_2 \in C$ such that:*
$\forall f_i \in c_2 \cap P \mid value_{c_2}(f_i) = value_P(f_i) \wedge \forall f_i \in c_1 \mid (\exists f_i \in c_2 \wedge value_{f_i}(c_2) = value_{f_i}(c_1)) \wedge conclusion(c_2) \neq conclusion(c_1)$

Also, as pointed out before, *critical questions* represent potential attacks that can defeat the conclusion of an argumentation-scheme.

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. Table 1 shows an example of a domain-case DC2 that a Global Regulator GR2 could retrieve to generate its recommended solution *"IR: Internal Redistribution"* for an overload problem in a FSS (based on the example of Section 3.2.3). This domain-case proposes redistributing resources inside the same physical machine, which is a *"BasicSolution"* solution that promotes *economy* (a value that promotes the lowest consumption of resources). The features that characterise the previously solved problem that DC2 represents are the following: service identifier, service current demand, virtual machines associated, physical resources associated to these virtual machines, and resources usage.

PROBLEM	Service	FSS
	Demand	SD1
	VMs	VM1, VM2, VM3
	Resources	VM1R, VM2R, VM3R
	Resources Usage	VM1RU, VM2RU, VM3RU
SOLUTION	Description	Internal Redistribution
	Solution Type	Basic Solution
	Value	Economy

Table 1
Domain-case Example

Next, we provide an example of an argumentation-scheme that changes the value preference order of Global Regulator in case of an overload in a service (inspired by Waltons's *argument for an exceptional case* [46]):

Major Premise: if the case of x is an exception, then the value preference order can be waived and changed by economy<quality 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 economy<quality in the case of network overload.

Thus, this scheme will change the social context of the Global Regulator in charge of negotiating the redistribution of resources to deal with the overload problem. Concretely, the Global Regulator goes to prefer more quality solutions instead of solutions that promote less consumption of resources. Therefore, the Global Regulator GR2 could use this scheme to generate its position and decide that the domain-case of Table 1 is not appropriate to solve its current problem and use another domain-case that supports its new social context.

However, the structure of argument-cases is generic and domain-independent. Table 2 shows an argument-case that represents a support argument SA1 in the context of the example of Section 3.2.3.

Argument-cases store the information about a previous argument that an agent posed in a specific step of a dialogue with other agents. Therefore, 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.

PROBLEM	Domain Context	Premises $\cup$ {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 = $\emptyset$	
		Counter Examples = DC4	
JUSTIFICATION	Cases = ...		
	Dialogue Graph = ...		

Table 2
Argument-case Example

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.

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 redistribution of resources.

For instance, the argument-case of Table 2 stores information about a support argument that a Global Regulator GR1 provided in the past to the Global Regulator GR2 to justify its position *"Instantiate new VM"* for an overload problem in the Virtual Machine VM1. This position promotes the value preferred by GR1 (e.g. quality of the service (QU)) and GR1 and GR2 had an authorisation dependency relation between them. As shown, the argument was unaccepted at the end of an argumentation process where it was proposed, since a counter-example DC4 was able to defeat it.

3.2.2. Real-Time Argumentation Process

In order to temporal bound the argumentation process that Global Regulators follow, we have used the approach presented in [30] (represented in Figure 4). Thus, the process can be divided into three phases: generation and selection of positions, where the Global Regulator generates its potential positions and select the best one to propose; evaluation of positions, where the Global Regulator evaluates the positions generated by other Global Regulators; and finally, argument management, where Global Regulators can either defend their positions if they are attacked or else, attack other different positions proposed by their peers. To defend their positions, Global Regulators have to evaluate the attack arguments received and generate and select the best support argument to propose. To attack different positions, Global Regulators have to ask the agent that they want to attack for providing them with a support argument for the position to attack. Then, Global Regulators can generate and select the best argument to attack the support argument provided.

As represented in Figure 4, the function *generatePositions* generates the k first positions by using the algorithm *generatePositions*. In the worst case, the function has to check the whole case-base of domain-cases, incurring a temporal cost $O(n)$ (where n is the number of cases stored in the domain-cases case-base) for finding and extracting similar domain-cases. Then, for each domain-case extracted, the function reuses its solution to generate a position. Again, in the worst case (in the temporal sense), we can generate n positions. Therefore, the *generatePositions* function has an asymptotic temporal cost of $O(n^2)$. Also, the *computeSF* is a function that computes a support factor for each position. With this factor, agents are able to evaluate the expected utility of a current position or argument in view of how "persuasive" it was in a similar argumentation process in the past. As criteria for making this evaluation, we consider different parameters computed from the elements of those argument-cases stored in the agent's case-base that represent argumentation experiences that are similar to the current one. In the worst case, this function has to check the whole case-base of argument-cases twice. Therefore, this process has an asymptotic temporal cost of $O(m^2)$, where m is the number of cases stored in the argument-cases case-base. For the rest of functions of Figure 4, we assume a constant temporal cost that is negligible in terms of the total temporal cost of the argumentation process. See [30] for more details on this temporal cost evaluation.

Summarising, the asymptotic cost to execute the process to generate and select positions in the worst case is $O(n^4 * m^3)$ where n is the number of cases stored in the domain-cases case-base and m is the number of cases stored in argument-cases case-base of the Global Regulator. Therefore, to be able to make a decision about the best solution to apply for the service overload problem in a time restricted by the SLA contracted, the maximum number of cases in the domain-cases and the argument-cases case-base must be known and fixed in advance. In this way, if the contents of the Global Regulator's case-bases contain the necessary information to generate and select an appropriate position, the agent will be able to generate at least one solution on time (its position). In this sense, as shown in Figure 4 the position generation and selection phase is mandatory and the other two phases (position evaluation and argument management) are optional and executed while the agent has still time to perform its temporal bounded reasoning process.

Once the process to generate and select positions has finished, agents of our framework can either receive attacks to their positions or decide to challenge the positions of other agents. Thus, agents are able to evaluate its position with regard to other positions. The first step to evaluate an agent's position is to check if it is consistent with the positions of other agents (they are the same). Agents do not attack consistent positions, but they can still generate support arguments if their own positions are challenged. Another possibility arises when a proponent agent has been able to generate the same position as another agent, but it has selected a different position as more suitable to propose. In that case, the proponent would accept the opponent's position if the latter has a power relation over the proponent and would try to attack the opponent's position otherwise. Finally, if the opponent's position is not in the set of positions generated by the proponent and the opponent does not have a power or authorisation relation over the proponent, the proponent can try to generate an argument to attack the opponent's position. Otherwise, it accepts the opponent's position.

In our real-time argumentation scenario the number of Global Regulators engaged in the argumentation process cannot be determined a priori. Therefore,

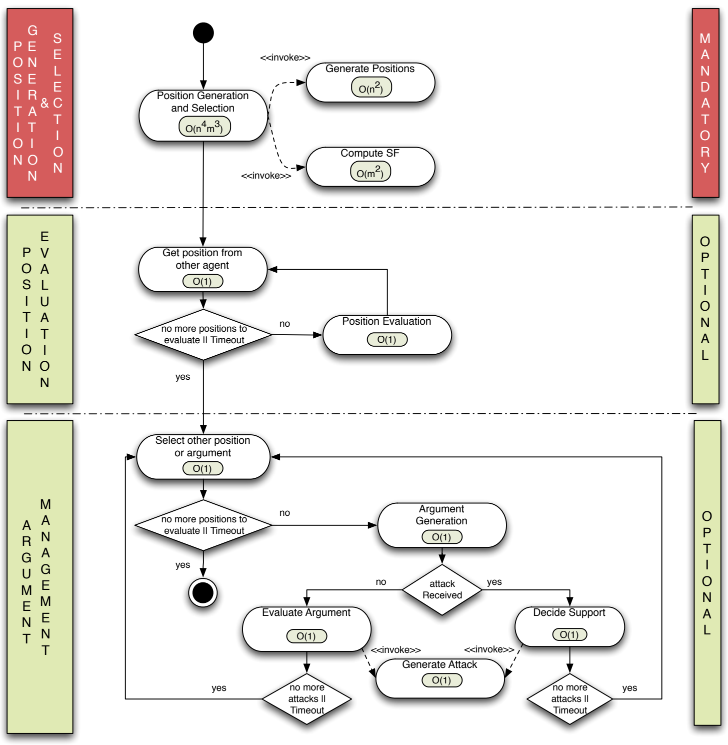

Fig. 4. Real-Time Argumentation Process

the temporal cost of evaluating the positions of other agents cannot be bounded. Thus, this phase has been implemented as an anytime process. The agent has a maximum time (a *deadline* based on the SLA of the service to provide) to evaluate all positions generated by other agents. Once this deadline is exceeded, the agent stops the evaluation and proceeds to the next phase of its reasoning process.

In the argument management process, agents generate arguments when they are asked to justify their positions (support arguments) or when they attack others' positions or arguments (attack arguments). A support argument has a support set that consists of the set premises that describe the problem and of any of the knowledge resources used to generate the position to justify (domain-cases and argument-cases). Attack arguments are generated when the proponent of a position provides an argument to justify it and an opponent wants to challenge the position or else, when an opponent wants to attack the argument of a proponent. If the agent receives an attack, it must evaluate its current argument in view of the incoming attack argument. The attack arguments that an agent can generate depend on the elements of the support set of the attacked argument. On one hand, if the support set includes a set of premises, the agent can generate an attack argument with a distinguishing premise. On the other hand, if the support set includes a domain-case or an argument-case, the agent can check its case-bases to find counter-examples to generate the attack. As for the case of generating positions, agents can generate several attack arguments. Then, to select the best attack argument to put forward in a specific step of the real-time agreement process, the agent uses the information of its argument-cases case-base and selects such one that is expected to have higher persuasive power in view of the agent's previous experiences.

Similarly to the evaluation process, we cannot estimate the number of arguments that a Global Regulator has to evaluate and the underlying attacks that this agent can receive or generate (what is called argument management in Figure 4). Therefore, we use here an anytime process by which the Global Regulator has been assigned a deadline to perform all interactions with the rest of the agents. When this deadline is finished, the argument management phase ends and the agent must provide its final solution. Therefore, in the worst case the agent has not enough time to argue.

Otherwise, if the allowed time is large enough, Global Regulators can make an intensive use of its domain and argumentation knowledge and engage in the argument management phase proposing and defending all positions that they are able to generate.

The argumentation process finishes when the deadline specified is reached or no new positions or arguments are proposed after a certain time. Then, the Global Regulator that started the argumentation dialogue must report to the Local Manager the final solution agreed. Thus, if several positions and their underlying support arguments remain acceptable, the most frequently proposed position is selected as the final solution to propose. In case of draw, a random choice is made.

3.2.3. Example

In order to illustrate how our argumentation framework can be applied to manage the service overload problem, this section describes the real-time argumentation process among several Global Regulators (GRs) by means of an example.

First, we assume that the SLA Manager has had a previous negotiation process with the client of the +Cloud platform and, as a result, a SLA has been signed for a specific service contracted with the Cloud provider (a file storage service, FSS, for instance). The process starts when a Service Demand Monitor notices an overload in the FSS that it controls. Then, the Service Supervisor in charge of the virtual machines that offer this service sends the load information about the resources associated to these virtual machines to the Local Manager of its physical machine. 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 has to make the best decision to redistribute its virtual and physical resources to cope with this overload problem and provide the service within a maximum time (*deadline*) specified in the SLA.

In this example, the Local Manager decides that the redistribution of resources to deal with the peak of demand implies an inter-machine configuration. Therefore, it transfers the information to the GR of the same physical machine, and then, the GR starts an argumentation process with its peers in other physical machines to decide which is the best solution to propose to its service overload problem. GRs are connected among them and that they are able to check the positions proposed by other GRs. Furthermore, all agents are collaborative and follow the common objective of providing the best solution by making the most of their individual experiences. A GR 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 way, the more *justified* position is selected as the solution to propose to the Service Supervisor, which in turn will transfer this recommendation to the Local Manager.

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

– **Quality:** agents that promote this value will select those solutions that improve the quality of the service[1].
– **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 VO, although the scenario could be extended to consider agents that belong to different VOs (for instance, those which group together agents that manage the same type of services). In addition, agents can play different roles from the ones defined in +Cloud and even act as representatives of a group (e.g in this agreement process, GRs act as representatives of 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, two agents that play the role of Global Regulators, GR2 and GR3, are arguing with the Global Regulator that started the agreement process, GR1, to decide which is the best redistribution 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 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).

[1]The notion of quality of service can have different meanings depending on the SLA contracted (e.g. response time, client's satisfaction level, etc.)

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 to solve in this example 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 and the *position generation & selection phase* starts. 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. To generate positions, each GR retrieves from its domain case-base those cases that match with the specification of the current problem and generates its solution (or a list of potential solutions) by reusing the solution(s) applied to the retrieved cases. Thus, with the set of retrieved cases GRs could provide different solutions for the same problem. Then, each GR can use its case-base of argument-cases to select the best position to propose, computing its support factor.

GRs have bounded the size of their case-bases to be able to ensure that they can meet the deadline to generate and select positions. Figure 5 presents a potential domain-case that GR2 could retrieve to generate its recommended solution *"IR: Internal Redistribution"* (since it has deemed it as similar enough to the current problem). This DC2 proposes redistributing resources inside the same physical machine, which is a *"BasicSolution"* solution that promotes *economy*. Since GR2 has been able to generate a solution on time, it will engage in the argumentation process proposing its position, which entails to recommend the solution generated.

In the case of GR1 and GR3, they have found the domain-case *DC3* (each one in its own domain-cases case-base), which proposes an alternative solution (e.g *"NVM: Instantiate a new VM"*) that promotes *quality* and has the type of solution *"EasySolution"*. Figure 6 shows an example of this domain-case retrieved from the case-base of GR3. Again, since GR1 and GR3 have been able to generate a solution on time, they will engage in the argumentation process proposing its position. Note that if GR1 would not be able to generate its own position, it could still participate in the argumentation process as *initiator* and wait for the final solution agreed between the other agents.

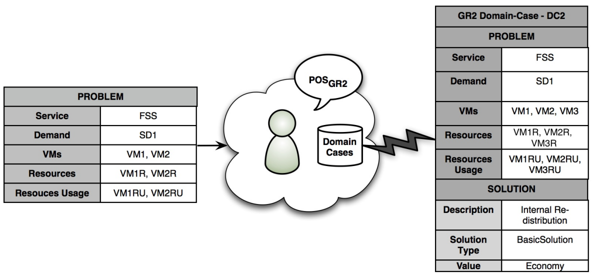

Fig. 5. Domain-Case DC2

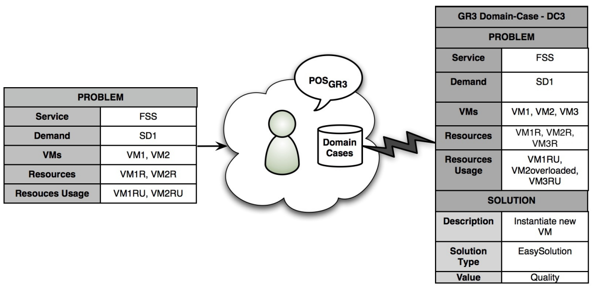

Fig. 6. Domain-Case DC3

Once the agents have proposed their positions, GR1 has to decide which between them could be the best solution to apply. Therefore, while the deadline to provide a solution is not met, it starts the *position evaluation phase* for each position proposed by the other GRs. In this example, since GR3 has proposed the same position as GR1, GR1 asks GR2 to provide an argument for supporting its position. Assuming that GR2 is willing to collaborate and there is still time to enter the *argument management phase*, it can put forward the following support argument for its position:

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

where the support set includes the *premises* of the problem description and the domain-case used by GR2 (DC2) to generate its position.

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. Therefore, if asked for supporting its position by GR2, GR3 could generate the following argument:

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

However, our concrete domain application centralises the argumentation process in the initiator of the dialogue. Then, GR1 has to evaluate the argument of GR2 (since GR3 is considered a supporter of GR1's position). Now, let us suppose that GR1 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 (see example in Section 3.2.1). Thus, this scheme will change the social context of the attack argument that the GR1 is going to create. As the support set of *SAGR*2 contains a domain-case, GR1 will try to propose a counter-example or a distinguishing premise for this case.

Thus, GR1 will check its case-base of domain-cases to find counter-examples for DC2. Suppose that GR1 finds two counter-examples for DC2 (DC1a and DC1b) which subsume the problem description of each of DC2 (but also including the new premise that states the overload of VM1), but providing different solutions (e.g. "NVM = Instantiate a new VM" and "ER = External Redistribution"). It could, therefore, generate the following attack arguments:

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

to undercut SAGR2 by attacking its support element DC2 with the counter-example DC1a, which promotes quality (QU) (as stated by S1).

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

to undercut SAGR2 by attacking its support element DC2 with the counter-example DC1b, which entails a *complex solution* that migrates the overloaded VM1 to another physical machine with enough resources to host it and promotes economy (EC) (assuming that the cost of migrating an existing VM is lower than instantiating a new one).

The argument management phase continues while GR1 has time to generate more attacks to GR2. Then, GR1 will try to find distinguishing premises and will check that the problem description of the domain-case DC2 matches the extended description of the problem (the original description plus the new premise VM1overloaded). In this example, GR1 realises that DC2 does not match with the extended description and generates a new attack argument to GR2:

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

to undercut SAGR2 by attacking its support element DC2 with the distinguishing premise *VM1overloaded*. Here, we assume that by attacking the argument of GR2 with a distinguishing premise, GR1 supports its initial position "'NVM" and then promotes quality (QU).

Now, GR1 has to select the argument that it will pose to attack the position of GR2. 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 still has 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 an argumentation process where it was also in the exceptional situation of an overload in the VM1 (Table 2 of Section 3.2.1 shows an example). However, a counter-example that also included this 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. Thus, GR1 will 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 generate more support arguments for its position. In case that it cannot generate more support argument or, GR2 would have to withdraw its position pos_GR2. If no more positions or arguments are generated (or if the deadline is met during the argumentation process), GR1 (and GR3) position would be selected as the best solution to deal with the overload problem of service FSS.

If the deadline specified in the SLA is never met, the argumentation process finishes when no new positions or arguments are proposed after a certain time. Then, the GR that initiated the argumentation 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 GRs during the argumentation dialogue. Finally, once a position is selected as the outcome of the argumentation process, each GR that participated in the process sends it to the Local Manager of its physical machine. In its turn, the GR that initiated the argumentation process will convey the agreed solution to the Global Supervisor of its VO. Finally, the Local Managers of the physical machines implied in the recommended solution (supervised by the Global Supervisor of their VO) would start the process to implement it. This implementation could entail with further negotiations that are out of the scope of this paper. Also, at the end of the argumentation process, all GRs 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.

4. Evaluation

In this section, we evaluate our proposals with a real example based on the problem introduced in Section 3.2.3, where the argumentation framework solves an overload in a service demand and hence, a lost in the quality of service offered by the +Cloud environment. We assume that the +Cloud platform is run by a company that has signed a SLA with a customer that has contracted a File Storage Service (FSS). Among other terms, this SLA specifies a maximum acceptable time within the customer receives an answer for this service (a *deadline* to save, search and retrieve files from the virtual directory hosted in the platform). If the solution exceeds this time, the company can receive an economic penalisation in its contract with the customer, which is greater as much time from the contracted in the SLA is exceeded.

This evaluation is focused on the real-time agreement capabilities of +Cloud at the internal level, that is, to ensure the fulfilment of the SLA contracted when a peak of service demand occurs. The evaluation of the negotiation protocol that allows to have real-time agreements at the external level implies the participation of real customers and remains future work. Then, our main objective with the present evaluation is to assess the efficiency of the system that implements the real-time argumentation framework. In order to do that, we have run evaluation tests several times populating the +Cloud platform with two different types of Global Regulators:

1. agents that implement our original case-based argumentation framework [23], where no real-time considerations were taken into account in the agents' reasoning process to manage positions and arguments (noRT agents).
2. agents that implement the real-time case-based argumentation framework proposed in [30], which follow a temporal bounded CBR cycle (TB-CBR) to manage their positions and arguments (RT agents). These type of agents are able to provide answers within the specific time that the SLA requires.

In the evaluation, the FSS has been subjected to a stress test where the load is incremented lineally over time and the system has to adapt itself to provide the service without breaking its associated SLA. Therefore, the Global Regulators of +Cloud implement the argumentation framework to be able to argue and make the best decision in order to solve the overload problem taking into account the temporal restrictions. Concretely, the simulation has been performed by progressively launching a set of threads, each of them continuously sending requests to a specific webservice exposed by the FSS. In other words, at the beginning of the test, we started with just one thread and periodically one new thread was launched (each 0,25 seconds), up to a maximum of 40 threads in parallel. The average duration of the test is 30 seconds.

Each test was run several times with two different webservices (at the SaaS layer) that make use of the FSS (at the PaaS layer):

– *GetFolderContents* which is a simple method that returns the content (files and directories) of a specific folder.
– *Find* which is a complex and recursive method that looks for a pattern in a specific folder.

The +Cloud platform was reconfigured to the same initial settings before running each test. The characterization of FSS during the test is shown in Figure 7 and is described as follows:

– The FSS is deployed in two virtual machines (VM1 and VM2). The virtual machine template of this service has a virtual processor of 1 core, 512Mb of RAM memory. Tornado Phyton[2] is used as web server.
– Each virtual machine is deployed on a different physical server (VM1R y VM2R).
– VM1R and VM2R host a known number of virtual machines, and all the physical resources will be shared among them. We assume that the reallocation of the resources at the intra-machine level is not possible. The Cloud environment has an unknown number of physical servers. This is one of the main advantages of our approach, which allows to make decision with a partial knowledge of the system state. Thus, there is no centralized algorithm with a complete knowledge about the system configuration, which is unrealistic in large-scale Cloud systems.
– The files are stored in a SAN (Storage Area Network) based on GlusterFS[3].
– The load balancer of the FSS is allocated in an additional virtual machine VMB. This virtual machine has a processor of 1 core, 256Mb of RAM Memory and it uses a Nginx[4] as inverse proxy.
– Finally, the FSS makes use of a database to store elements of the virtual file system, such as relations between files, metadata, permissions, etc. This database is based on mongoDB[5] and it is hosted in an unknown number of virtual machines with a processor of 1 core and 512Mb of RAM.

As in the example of Section 3.2.3, the adaptation (in terms of implementing the resource reallocation) starts when a Service Demand Monitor notices an overload in a FSS that it controls. The Service Supervisor in charge of the virtual machines that offer this service 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 and virtual resources load. The load of the resources is extracted by means of the library *libvirt* or system calls. This extraction process entails a computational cost, since it implies the computation of changes on the resources demand (mainly cpu, memory and bandwith). However, the computational cost depends on the frequency in which the measures are taken. For example, if the measures are taken every 0,5 seconds, the CPU load average used in this process is high (approximately 20%). However, if the measures are take every 3 seconds, the load average is reduced to less than 3%. Taking into account that with a frequency of requests between 10 to 20 the cloud systems works properly, we can state that this resource consumption is acceptable. The memory load and the bandwidth are negligible, and the storage need depends on the size of the round-robin database.

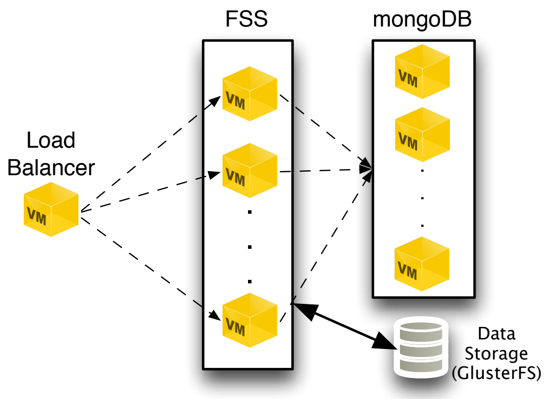

Fig. 7. FSS architecture during the test

With all this information, the Local Manager will make the best decision to redistribute its virtual and physical resources to cope with this overload problem. We assume that the Local Manager decides that the redistribution of resources implies an inter-machine configuration. Therefore, it notifies the problem to the Global Regulator of its same physical machine. Then, the Global Regulator starts an argumentation process with its peers in other physical machines of its VO to decide which is the best solution to solve the service overload. Also, this agreement process is temporal bounded to allow the FSS to be readapted without breaking the deadline specified in the SLA.

Each Global Regulator has its own knowledge resources to generate a solution for the problem reported. The argumentation module of each Global Regulator includes a Domain-CBR engine that makes queries to its domain-cases case-base and an Argumentation-CBR engine that makes queries to its argument-cases case-base. For the tests, a real database of 50 overload problems in the FSS solved in the past is used as domain knowledge. Despite the small size of this case-base, we have rather preferred to use ac-

[2]http://www.tornadoweb.org/
[3]ttp://www.gluster.org/
[4]http://nginx.com/
[5]http://www.mongodb.org/

tual data instead of a larger case-base with simulated data. The argument-cases case-bases of each agent are initially empty and populated with cases as the agents acquire argumentation experience during the experiments, reaching to a maximum number of 20 argument-cases that agents are able to learn with our experimental settings. In each test, a Global Regulator agent receives one problem to solve per run and contacts its peers to report them the problem to solve. We make the same assumptions as in Section 3.2.3: Global Regulators are able to communicate among them, they are collaborative and follow the common objective of providing the best solution by making the most of their individual experiences.

Figure 8 shows how the system behaves without any kind of adaptation and without observing any deadline. Therefore, as time passes, the number of concurrent threads making requests to the FSS service, and hence the service load, increases. Subsequently, the response time for both webservices also increases, specially for the *Find* webservice due to its higher complexity, until it stabilises. We use the average response time for both webservices in this test to fix the deadlines established in different SLAs in the following tests.

In the first test, the percentage of FSS requests that are served on time within the agreed SLA is evaluated. In the context of this evaluation we use a simplified version of a SLA, where the only commitment that must be fulfilled is the response time to provide the service in each request. Table 3 shows the percentage of times that different SLAs, which entail different deadlines to provide the service, are fulfilled by using RT agents, noRT agents and, also, without any type of adaptation (Global Regulators are not able to argue to solve problems). All results show the the average percentage of SLAs fulfilled including all the requests done during all phases within the test (from the load increment that overloads the service, the argumentation process, the reallocation of resources and, finally, the stabilisation of the service).

The results in Table 3 show that, firstly, the adaptation of the system is performed properly by allowing Global Regulators to argue, since the percentage of requests to the FSS that are served on time is much higher when the adaptation is performed, independently of the type of agents (RT or noRT) that are used in the argumentation process. Secondly if we compare the percentages between noRT and RT agents, the results show that the RT agents offer better solutions for the adaptation of the system than the noRT agents. Thus, this is the proof that validates our hypothesis that real-time argumentation enhances the performance of +Cloud when a service is overloaded during a peak of demand. Furthermore, as less restrictive the deadline established in the SLA is, as much percentage of requests can be served on time in all cases. However, if we go deeper in the data and the less and more restrictive SLAs of both webservices are excluded, the percentages of requests that are served on time are much higher with RT agents than with noRT agents (about 30 units in the *GetFolderContents* webservice and 10 in the *Find* webservice). Finally, as expected, the adaptation of the system when the overloaded webservice is simple, *GetFolderContents* in our case, gets better results than in the case of the more complex *Find* webservice (which consumes more processing resources).

Once the initial hypothesis is proven, our evaluation tests illustrate why RT agents get better adaptation results than noRT agents. For these tests, a deadline of 0.4s for the *GetFolderContents* webservice and of 1.8s for the *Find* webservice is specified in the SLA contracted for the FSS. On the one hand if the system runs with noRT agents and an readaptation is needed, the response time of the requests served after the adaptation of the system is not linear. Figure 9 shows how, although the average response time to provide the service decreases after the adaptation, the distribution of this time is scattered. This occurs since noRT agents are able to argue, but without observing any restriction on the deadline to provide the service. However, as shown in Figure 10, if the system runs with RT agents the argumentation process is temporal bounded and tries to fulfil the SLA agreed, resulting in a less scattered distribution of the response times.

Our proposal has been tested in different scenarios and with different services (FSS, OSS, etc.). It allows to reallocate the individual resources of an individual virtual machine, migrate it, or even, create or destroy the instances of a virtual machine. The real-time agreement mechanism presented in this paper works properly with a substantial number of virtual machines assigned to the same service (tested up to 12 virtual machines), but it can be scaled up to a higher number of instances, since all decisions regarding the redistribution of resources are taken in a decentralised way following a MAS approach.

5. Discussion

During the last years, Cloud Computing has emerged as a new paradigm for hosting and service provision

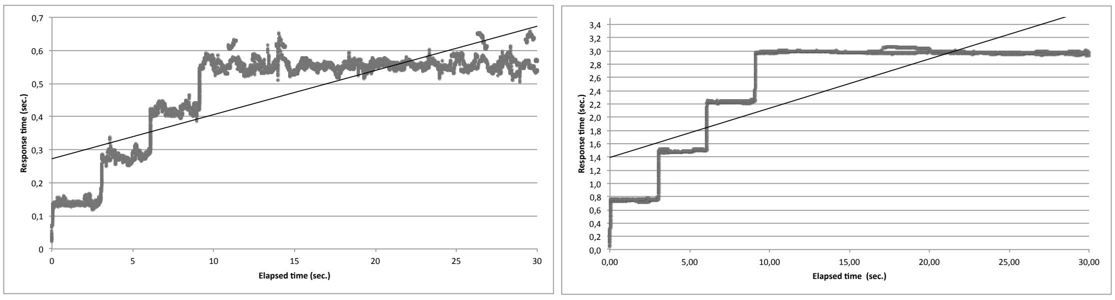

Fig. 8. Test performed without reallocation of the resources (Left: GetFolderContents; Right: Find)

Method getFolderContents

SLA	**no Adapt.**	**noRT**	**RT**
SLA = [Resp. time = 0,3s.]	20,32%	33,88%	75,39%
SLA = [Resp. time = 0,4s.]	21,44%	53,86%	82,33%
SLA = [Resp. time = 0,5s.]	31,52%	76,99%	87,99%
SLA = [Resp. time = 0,6s.]	96,34%	97,22%	99,07%

Method Find

SLA	**no Adapt.**	**noRT**	**RT**
SLA = [Resp. time = 1,4s.]	13,39%	39,56%	47,83%
SLA = [Resp. time = 1,6s.]	23,15%	59,50%	72,04%
SLA = [Resp. time = 1,8s.]	23,76%	69,40%	76,80%
SLA = [Resp. time = 2,0s.]	24,45%	74,45%	78,08%

Table 3

Percentage of FSS requests that are served with no Adaption, noRT agents and RT agents

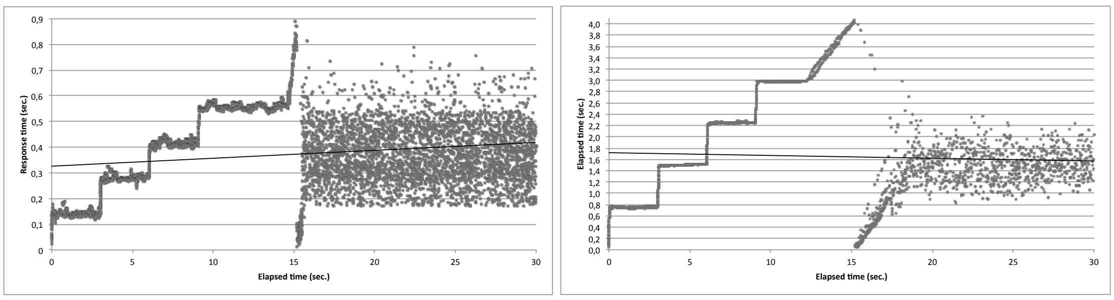

Fig. 9. Example of adaption using noRT agents (Left: GetFolderContents; Right: Find)

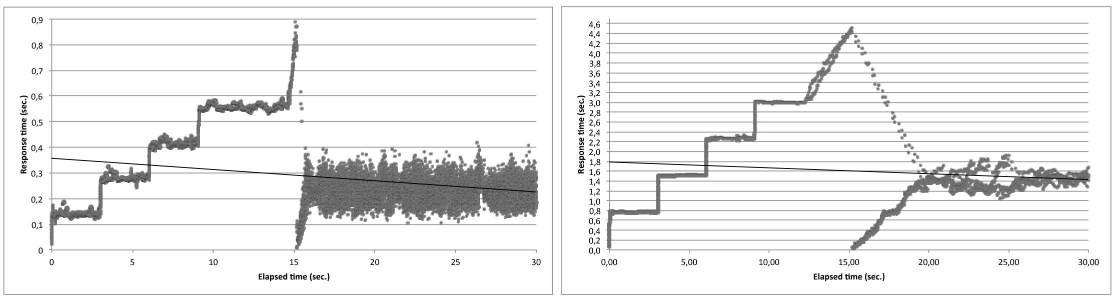

Fig. 10. Example of adaption using RT agents (Left: GetFolderContents, SLA=[resp. time:0.4s.]; Right: Find, SLA=[resp. time:1.8s.])

over the Internet. Cloud Computing tries to reduce the user requirements and to provide a dynamic resource management, where the resources increase only when there is a rise in service demand [8, 9]. The efforts on IT industry have been focused on the development of many Cloud platforms (Google Apps[6], Amazon

[6]http://www.google.com/apps

EC2[7]/S3[8], etc.) that try to offer different computational services at different levels (software, platform and/or infrastructure) following a Cloud approach. That is, offering services with the same level of quality independently of the demand [34]. Also, the number of open source platforms has been rapidly increased, but these majority of well-known open platforms tend to underscore their ability to provide elastic infrastructure services (through virtualised hardware), without taking high level services, such as platform and software, into account [34]. The competitive advantage associated with knowing and dominating this technology is the key to understand the rapid increase of a technology that as little as five years ago did not even exist. On the other hand, the efforts on the academia are a set of scattered research framed in many areas such as resources allocation [28], security [24], interoperability [12], etc. The majority of these works can be associated with previous computational paradigms (grid computing, cluster computing, etc.) or related technologies such as virtualisation that are being moved close to the Cloud Computing paradigm.

Although at first glance this paradigm appears to be simply a technological paradigm, reality shows that its rapid progression is primarily motivated by economic interests that surround the purely computational or technological characteristics. This type of platform will allow us to offer computational services following a model comparable to the concept proposed by Utility Computing [37], where computational services are offered in a similar way to how traditional public utilities (electricity, telephone, water, etc.) are. As a result, Cloud service providers will be able to offer their demanded services through the Internet with a pay-per-use model [13] [18]. This new business model throws the need to agree on the terms of the service between the Cloud provider and the client through a SLA. This has motivated the development of research work in this area [13] [3].

Recent research foresees the emergence of a new discipline called agent-based Cloud Computing systems that try to join both computational paradigms, Cloud Computing and MAS, to lead to an advanced computational environment based on the computational strength of the former and the intelligence and adaptation capabilities of the latter. Current literature reflects few references of studies that combine both agents and Cloud Computing paradigms. Most works in this area are focused either on the deployment of the system or the management of their resources. Within the first group, the state of the art reports MAS that mainly use the computational resources of the Cloud environment, in terms of computational strength to perform simulations, or as a persistence example [14]. In the second group, [39] highlights three subgroups of applications: (i) combination of resources among Cloud providers; (ii) planning and coordination of shared resources; and (iii) establishing contracts between users and Cloud service providers. As the authors point out, it is possible to find studies that develop a Cloud service using agents for different specific purposes [26][15]. Some notable examples of Cloud providers combining resources include works that share Cloud resources to offer Infrastructure as a Service (IaaS) [21][40]. Other works apply SLA to distribute services [39][4], but they are designed as individual services and not as a whole Cloud system. However, the main difference between these systems and our +Cloud platform is that they are not internally and completely managed by a MAS (i.e. the management of the virtual and physical resources in +Cloud is performed in a distributed way following the MAS paradigm).

In addition, MAS based on virtual organizations can enhance Cloud environments with advanced capabilities for automatic evolution and adaptation. A cloud environment can support multiple services, protocols, hardware platforms and operating systems. These environments are flexible and designed to handle the multiple requirements on services and usage of their users. By their very nature, virtual organizations of agents can support different types of agents running on different types of operating systems or platforms. In this way, a cloud infrastructure managed by agents can handle different technologies, and manage several sensors and actuators. Also, Cloud environments are dynamic. Thus, they must provide support for changing levels of network traffic and processing capabilities, and must be able to be configured in multiple ways to generate optimal results. In this sense, organizations of agents with deliberative capabilities can adapt processing components, re-distribute the workload and even add more nodes, virtual machines and/or computer systems to adapt the system to the needs of the network. Therefore, our work addresses research within this area and uses real-time agreements technology to reallocate resources within the system in order to cope with the changing demand.

Nowadays, most work reported in the reallocation of resources literature is focused on saving energy

[7]https://aws.amazon.com/ec2/
[8]http://aws.amazon.com/

[9][8][7], but few takes into account the need of fulfilling a SLA in the reallocation process [19]. Current main challenges in this area include resource modelling, resource offering and treatment, resource discovery and monitoring and the resource selection [17]. However, current algorithms to redistribute resources in Cloud systems present many weaknesses, such as the need of a heuristic, or a centralized or semicentralised approach [48].

Moreover, to incorporate real-time performance is one of the open issues that are still to be addressed in Cloud environments. The main efforts in this line have been focused on SaaS. Some approaches as Arithum[9] provide a real-time Cloud Computing platform and deliver pioneering real-time consumer and enterprise solutions. Some authors focus on specific real-time issues. Wei et al. [45] focus on multi-tenancy architecture, scheduling, paralleled computing and propose a framework for real-time service-oriented Cloud Computing. Specifically, they propose a real-time architecture which solves the new challenges in Cloud Computing. Liu et al. [29] introduce a utility accrual scheduling algorithm for real-time Cloud Computing services. The real-time tasks are scheduled non-preemptively with the objective of maximising the total utility. Yu et al. [49] proposed a task model that considers both the profit and penalty that a system may incur when executing a task. These contributions pave the way for an interesting new area of research in Cloud-based multi-agent systems, real-time and SLAs, but so far, there is only a limited number of proposals in the literature.

Argumentation-based agreement technologies, as a proficient research area within MAS-based agreement models, can also contribute towards the achievement of new challenges in agent-based Cloud Computing. In recent years, the community 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 [36].

Case-Based Reasoning is another research area where argumentation theory has produced a wide history of successful applications [22]. According to the CBR methodology, a new problem can be solved by searching in a case-base for similar precedents and adapting their solutions to fit the current problem. This reasoning methodology has a high resemblance with the way by which people argue about their positions (i.e. beliefs, action proposals, opinions, etc.), trying to justify them on the basis of past experiences. Many argumentation systems produce arguments by applying a set of inference rules [11][35][25], which require to elicit an explicit model of the domain. In the context of dynamic open MAS, this model is difficult to specify in advance. However, storing in cases the arguments that agents put forward and reuse later this information could be relatively simple.

The work done in the eighties and nineties about legal CBR fostered the argumentation research in the AI community [5][2]. Nowadays, the argumentation research in CBR continues being very active and in addition to our framework, some approaches that integrate CBR in MAS to help argumentation processes have already been proposed. However, these proposals tend to be highly domain-specific (e.g. persuasion [27], sensor networks [41]) or centralise the argumentation functionality in a mediator agent that manages the dialogue between the agents of the system, as in the ProClaim framework [43]. In addition, the agents of the *AMAL* case-based argumentation framework presented in [31] can also learn during the argumentation dialogue by storing in their case-cases the cases that they receive from other agents. However, by contrast with our proposal, they do not learn how to predict the expected acceptability of arguments, but only increase their own knowledge with the knowledge that other agents share with them.

Furthermore, as in our framework, argumentation schemes have been used to model argumentation dialogues between agents. The ProClaim case-based argumentation framework also uses argumentation schemes to represent generalised patterns of reasoning. These schemes are manually designed by experts (doctors) in the application domain of this model (organ transplantation) and characterise the space of possible arguments that agents can generate. This implies that agents have limited expressiveness, although the authors state that this decision has been taken for security reasons in the critical domains where the model operates. A similar approach was followed in [44], where agents dynamically construct alternative plans according to the information acquired during the dialogue and a set of argumentation schemes for deliberative dialogues. However, this is again a rule-based system that assumes the existence of a predefined set of inference rules based on these schemes.

However, the application of argumentation approaches to Cloud Computing is a new research challenge, specially if the process entails real-time constraints, as

[9]http://www.arithum.com/

our work presents. In this work, we deal with one of the main challenges for agent-based solutions to Cloud software infrastructure, which states the advantages of using agents to create intelligent and flexible Cloud services [42]. These include Service Level Agreements (SLAs) based on real-time agreement technologies and load-balancing services based on MAS cooperation.

Acknowledgments

This work is supported by the Spanish government grants CONSOLIDER-INGENIO 2010 CSD2007-00022, TIN2011-27652-C03-01, and TIN2012-36586-C03-01.

Acronym	Description
A	Authorisation (agreement dependency relation)
C	Charity (agreement dependency relation)
DC	Domain Case
EC	Economy (agreement objective)
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	Multi-Agent System
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

Table 4
List of Acronyms

References

[1] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). http://www.ogf.org/documents/GFD.107.pdf, 2007.

[2] V. Aleven and K. D. Ashley. Teaching case-based argumentation through a model and examples, empirical evaluation of an intelligent learning environment. In *Artificial Intelligence in Education, AIED-97*, volume 39 of *Frontiers in Artificial Intelligence and Applications*, pages 87–94. IOS Press, 1997.

[3] M. Alhamad, W. Perth, T. Dillon, and E. Chang. Conceptual sla framework for cloud computing. In *4th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*, pages 606–610. IEEE Press, 2010.

[4] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[5] K. D. Ashley. Reasoning with cases and hypotheticals in hypo. *International Journal of Man-Machine Studies*, 34:753–796, 1991.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *9th ACM Symposium on Operating Systems Principles (SOSP-03)*, pages 164–177. ACM Press, 2003.

[7] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755–768, 2012.

[8] A. Beloglazov and R. Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 577–578. IEEE Computer Society, 2010.

[9] A. Beloglazov and R. Buyya. Energy efficient resource management in virtualized cloud data centers. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 826–831. IEEE Computer Society, 2010.

[10] T. Bench-Capon and G. Sartor. A model of legal reasoning with cases incorporating theories and values. *Artificial Intelligence*, 150(1-2):97–143, 2003.

[11] T. J. Bench-Capon. Specification and implementation of toulmin dialogue game. In *International Conferences on Legal Knowledge and Information Systems, JURIX-98*, Frontiers of Artificial Intelligence and Applications, pages 5–20. IOS Press, 1998.

[12] R. Buyya, R. Ranjan, and R. N. Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31. Springer-Verlag, 2010.

[13] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *Department of Computer Science and Software Engineering (CSSE), The University of Melbourne, Australia. He*, pages 10–1016, 2008.

[14] C. Chen and K. Wang. Cloud Computing for agent-based urban transportation systems. *IEEE Intelligent Systems*, 26:73–79, 2011.

[15] Y. Y. Cheng, M. Low, S. Zhou, W. Cai, and C. S. Choo. Evolving agent-based simulations in the clouds. In *3rd International Workshop on Advanced Computational Intelligence (IWACI)*, pages 244–249, 2010.

[16] F. Dignum and H. Weigand. Communication and Deontic Logic. In R. Wieringa and R. Feenstra, editors, *Information Systems - Correctness and Reusability. Selected papers from the IS-CORE Workshop*, pages 242–260. World Scientific Publishing Co., 1995.

[17] P. T. Endo, A. V. de Almeida Palhares, N. N. Pereira, G. E. Goncalves, D. Sadok, J. Kelner, and J. Mangs. Resource allocation for distributed cloud: concepts and research challenges. *Network*, 25(4):42–46, 2010.

[18] H. Erdogmus. Cloud computing: Does nirvana hide behind the nebula? *IEEE Software*, 26(2):4–6, 2009.

[19] J. O. Fitó, I. Goiri, and J. Guitart. SLA-driven elastic cloud hosting provider. In *18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 111–118. IEEE Computer Society, 2010.

[20] R. Fuentes-Fernández, S. Hassan, J. Pavón, J. M. Galán, and A. López-Paredes. Metamodels for role-driven agent-based modelling. *Computational and Mathematical Organization Theory*, 18(1):91–112, 2012.

[21] R. Grewa and P. Pateriya. A rule-based approach for effective resource provisioning in hybrid cloud environment. *International Journal of Computer Science and Informatics*, 4:101–106, 2012.

[22] S. Heras, V. Botti, and V. Julián. Challenges for a CBR framework for argumentation in open MAS. *Knowledge Engineering Review*, 24(4):327–352, 2009.

[23] S. Heras, J. Jordán, V. Botti, and V. Julián. Argue to agree: a case-based argumentation approach. *International Journal of Approximate Reasoning*, 54(1):82–108, 2013.

[24] M. Jensen, J. Schwenk, N. Gruschka, and L. Iacono. On technical security issues in cloud computing. In *IEEE International Conference on Cloud Computing*, pages 109 –116. IEEE Press, 2009.

[25] A. Kakas, N. Maudet, and P. Moraitis. Modular Representation of Agent Interaction Rules through Argumentation. *Autonomous Agents and Multi-Agent Systems*, 11:189–206, 2005.

[26] M. J. Kim, H. G. Yoon, and H. K. Lee. MAV: An intelligent Multi-agent model based on Cloud computing for resource virtualization. In *Computers, Networks, Systems, and Industrial Engineering*, volume 365 of *Studies in Computational Intelligence*, pages 99–111. Springer, 2011.

[27] S. Kraus, K. Sycara, and A. Evenchik. Reaching agreements through argumentation: A logical model and implementation. *Artificial Intelligence*, 104:1–69, 1998.

[28] W.-Y. Lin, G.-Y. Lin, and H.-Y. Wei. Dynamic auction mechanism for cloud resource allocation. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 591–592, Washington, DC, USA, 2010. IEEE Computer Society.

[29] S. Liu, G. Quan, and S. Ren. On-line scheduling of real-time services for cloud computing. In *6th World Congress on Services*, SERVICES '10, pages 459–464. IEEE Computer Society, 2010.

[30] M. Navarro, S. Heras, and V. Julián. Towards real-time agreements. *Expert Systems with Applications*, page In Press, 2013.

[31] S. Ontañón and E. Plaza. An Argumentation Framework for Learning, Information Exchange, and Joint-Deliberation in Multi-Agent Systems. *Multiagent and Grid Systems Journal*, 7(95-108):IOS Press, 2011.

[32] J. Palanca, M. Navarro, A. García-Fornes, and V. Julián. Deadline prediction scheduling based on benefits. *Future Generation Computer Systems*, 29(1):61–73, 2013.

[33] C. Pautasso, O. Zimmermann, and F. Leymann. Restful web services vs. "big"' web services: making the right architectural decision. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 805–814, New York, NY, USA, 2008. ACM.

[34] J. Peng, X. Zhang, Z. Lei, B. Zhang, W. Zhang, and Q. Li. Comparison of several cloud computing platforms. In *2nd International Symposium on Information Science and Engineering*, ISISE '09, pages 23–27. IEEE Computer Society, 2009.

[35] H. Prakken and G. Sartor. Modelling reasoning with precedents in a formal dialogue game. *Artificial Intelligence and Law*, 6:231–287, 1998.

[36] I. Rahwan and G. Simari, editors. *Argumentation in Artificial Intelligence*. Springer, 2009.

[37] J. W. Ross and G. Westerman. Preparing for utility computing: The role of it architecture and relationship management. *IBM Systems Journal*, 43(1):5–19, January 2004.

[38] H. E. Schaffer. X as a service, cloud computing, and the need for good judgment. *IT Professional*, 11(5):4–5, 2009.

[39] K. M. Sim. Agent-based cloud commerce. In *IEEE International Conference on Industrial Engineering and Engineering Management*, pages 717–721. IEEE Press, 2009.

[40] A. Singh and M. Malhotra. Agent based framework for scalability in cloud computting. *International Journal of Computer Science and Engineering Technology*, pages 41–45, April 2012.

[41] L.-K. Soh and C. Tsatsoulis. A real-time negotiation model and a multi-agent sensor network implementation. *Autonomous Agents and Multi-Agent Systems*, 11(3):215–271, 2005.

[42] D. Talia. Clouds meet agents: Toward intelligent cloud services. *IEEE Internet Computing*, 16(2):78–81, 2012.

[43] P. Tolchinsky, S. Modgil, K. Atkinson, P. McBurney, and U. Cortés. Deliberation dialogues for reasoning about safety critical actions. *Autonomous Agents and Multi-Agent Systems*, In Press, 2011.

[44] A. Toniolo, T. Norman, and K. Sycara. An empirical study of argumentation schemes in deliberative dialogue. In *20th European Conference on Artificial Intelligence, ECAI-12*, number 242 in Frontiers in Artificial Intelligence and Applications, pages 756–761. IOS Press, 2012.

[45] W.-T. Tsai, Q. Shao, X. Sun, and J. Elston. Real-time service-oriented cloud computing. In *IEEE 6th World Congress on Services, SERVICES'10*, pages 473–478. IEEE Press, 2010.

[46] D. Walton, C. Reed, and F. Macagno. *Argumentation Schemes*. Cambridge University Press, 2008.

[47] L. Wang, J. Tao, M. Kunze, A. Castellanos, D. Kramer, and W. Karl. Scientific cloud computing: Early definition and experience. In *10th IEEE International Conference on High Performance Computing and Communications (HPCC-08)*, pages 825–830. IEEE Press, 2008.

[48] Y. O. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pages 91–98. IEEE Computer Society, 2010.

[49] Y. Yu, S. Ren, N. Chen, and X. Wang. Profit and penalty aware (pp-aware) scheduling for tasks with variable task execution time. In *ACM Symposium on Applied Computing*, SAC '10, pages 334–339. ACM, 2010.