Scheduling Service Oriented Workflows Inside Clouds Using an Adaptive Agent Based Approach

Marc Eduard Frîncu

1 Introduction

In recent years Cloud Computing (CC) emerged as a leading solution in the field of Distributed Computing (DC). In contrast, Grid Computing lacked the open-world vision of overcoming some fundamental problems including transparent and easy access to resources, licensing or political issues, lack of virtualization support or to complicated to use architectures and end-user tools.

Clouds have emerged as a main choice for service vendors mostly due to their support for virtualization and service oriented approach. Inside clouds almost everything can be offered as a service. This has led to the appearance of several paradigms including Software as a Service (SaaS), Infrastructure-as-a-Service (IaaS) or Platform-as-a-Service (PaaS).

As more users begin to use clouds for storing or executing their applications these systems become susceptible to workload related issues. The problem is even harder when considering complex tasks which require accessing services provided by different cloud vendors (see Fig. 1) each with their own internal policies. Selecting the optimal /fastest service for a specific task becomes in this case an important problem as sometimes users are paying for their time spent using the underlying services.

Consequently scheduling tasks on services becomes even more difficult as inside cloud environments each member uses its own policies and is not obligated to adhere to outside rules. We end up with a bundle of services from various providers that need to be orchestrated together in order to produce the desired outcome inside a given time interval. Keeping the execution inside this interval minimizes production and client costs. As service selection requires some negotiation between providers one of the simplest and straightforward solutions is to use distributed agents that play the roles of service providers and clients.

Marc Eduard Frîncu
Institute e-Austria, Blvd. Vasile Parvan No 4 300223, Room 045B, Timisoara, Romania e-mail: mfrincu@info.uvt.ro

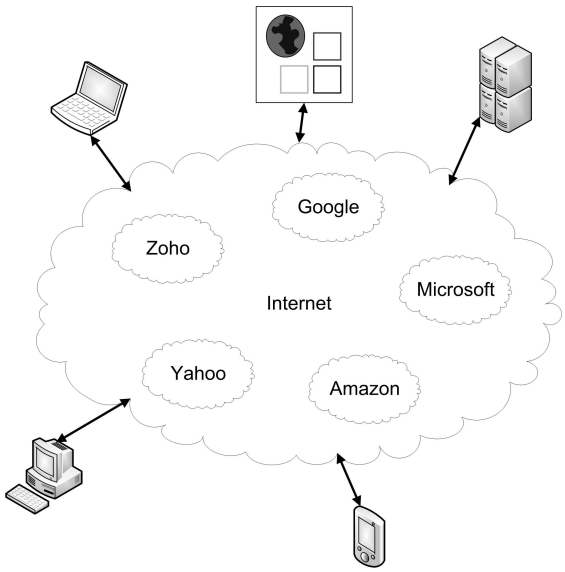

Fig. 1 Connecting devices to some of the existing clouds

This paper presents an agent based approach to the problem of task scheduling inside clouds. Two major problems are dealt with: finding cloud resources and orchestrating services from different cloud vendors towards solving a common goal. Certain deadline and cost constrains are assumed to exist. Even though the emphasis is put on workflow tasks, independent tasks can also be handled. Towards this aim we first present some solutions to the problem of task scheduling inside clouds. Then we present some issues regarding task scheduling inside Service Oriented Environments (SOE) together with some details on workflow scheduling. A detailed overview on a distributed agent based scheduling platform architecture capable of adapting itself to resource changes is also given. Finally a concrete experimental prototype and some conclusions are presented.

2 Related Work on DS Scheduling

Lot of work has been carried out in what concerns task scheduling inside Distributed Systems (DSs). This work can be divided into specialized Scheduling Algorithms (SAs) for clouds and Resource Management Systems (RMS) which discover services and allocate tasks to them. In what follows we briefly present some of the main work concerning both SAs for CC and RMS for DSs.

Concerning the development of efficient SAs for DSs, nature has proven to be a good place of inspiration. Recent papers such as [30], [41] try to cope with the problem of task scheduling by offering meta-heuristics inspired from behavioral patterns observed in ant colonies. This technique also called Ant Colony Optimiza-

tion (ACO) relies on the fact that ants inside a colony act as independent agents which try to find the best available resource inside their space by using global search techniques. Each time such an agent finds a resource better than the already existing one it marks the path to it by using pheromones. These attract other ants which start using the resource until a better one is found.

In [5] an ACO based approach for initiating the service load distribution inside clouds has been proposed. Simulated results on Google Application Engine [27] and Microsoft Live Mesh [32] have shown a slight improvement in the throughput of cloud services when using the proposed modified ACO algorithm.

The biggest disadvantage ACO has over other approaches is that it is not very effective when dynamic scheduling is considered. The reason for this is that rescheduling requires a lot of time until an optimal scenario is reached through intensive training given by multiple iterations. Because DSs are both unpredictable and heterogeneous each time a change is noticed the entire system needs to be trained again. This process which could last several hours. The large retraining time interval is not acceptable when tasks are scheduled under deadline constraints as the scheduling could take longer than the actual task execution. An improvement on this might be given by mixing the time consuming global search with local search when minor changes occur inside the DS. However defining the notion of minor changes is still an open issue.

Paper [26] deals with High Performance Computing (HPC) task scheduling inside clouds. Energy consumption is important both in what concerns the user costs and in relation to the carbon emissions. The proposed meta-scheduler takes into consideration factors such as energy costs, carbon emission rate, CPU efficiency and resource workflows when selecting an appropriate data center belonging to a cloud provider. The designed energy based scheduling heuristics shows a significant increase in energy savings compared with other policies.

Most of the work concerning RMSs has evolved around the assumption of applying them onto grids and not clouds. This can be explained by two facts. The first one is that there are many similarities between a cloud and a grid and RMS developed for one type could also work well on the other. The second one is related with age, and as grids emerged earlier than clouds most of the solutions have been developed for the former. Nonetheless several of the grid oriented RMSs could be adapted to work for clouds too.

One example is represented by the CloudScheduler [17]. It allows users to set up a Virtual Machine (VM) and submit jobs to a Condor [48] pool. The VM will be replicated on machines and used as container for executing the jobs.

In what follows we present some of the most known examples of RMS for DSs in general.

Notable examples include the Globus-GRAM [20], Nim- rod/G [8], Condor [48], Legion [16], NetSolve [14] and others. Many of these solutions use fixed query engines to discover and publish resources and do not rely on the advantages offered by distributed agents.

The ARMS [9] system represents an example of agent based RMS. It uses PACE [10] for application performance predictions which are later used as inputs to the scheduling mechanism.

In paper [45] a multi-site agent based scheduling approach consisting of two distinct decision levels one global and the other local is presented. Each of these levels has a predictive and a reactive component for dealing with workload distribution and for reacting to changes in the workloads.

Paper [11] presents a grid load balancing approach by combining both intelligent agents and multi-agent approaches. Each existing agent is responsible for handling task scheduling over multiple resources within a grid. As in [45] there also exists a hierarchy of agents which cooperate with each other in a peer to peer manner towards a common goal of finding new resources for their tasks. This hierarchy is composed of a broker, several coordinators and simple agents. By using evolutionary processes the SAs are able to cope with changes in the number of tasks or resources.

Nimrod/G uses agents [2] to handle the setup of the running environment, the transport of the task to the site, its execution and the return of the result to the client. Agents can also record information acquired during task execution as CPU time, memory consumption etc.

Paper [46] proposes a system which can automatically select from various negotiation models, protocols or strategies the best one for the current computational needs and changes in resource environment. It does this by solving two main issues DS have to dealt with [9]: scalability and adaptability. The work carried in [46] creates an architecture which uses several specialized agents for applications, resources, yellow pages and jobs. Job agents for example are responsible for handling a job since its submission and until its execution and their lifespan are restricted to that interval. The framework offers several negotiation models between job and resource agents including contract net protocol, auction and game theory based strategies.

AppLeS (Application-Level Scheduling) [7], [15] is an example of a methodology for adaptive scheduling also relying on agents. Applications using AppLeS share a common architecture and are scheduled adaptively by a customized scheduling agent. The agent follows several well established steps in order to obtain a schedule for an application: resource discovery, resource selection, schedule selection, application execution and schedule adaptation.

3 Scheduling Issues Inside Service Oriented Environments

Scheduling tasks inside SOE such as clouds is a particular difficult problem as there are several issues that need to be dealt with. These include: estimating task runtimes and transfer costs; service discovering and selection; negotiation between clients and different cloud vendors; and trust between involved parties. In what follows we address each of these problems separately.

3.1 Estimating Task Runtimes and Transfer Costs

Many SAs require some sort of user estimates in order to provide improved scheduling solutions. The estimates are either user estimated or generated by using methods involving code profiling [31], statistical determination of execution times [18], linear regression [33] or task templating [4], [47]. When applied to SOE these methods come both with advantages and disadvantages as it is shown in the next paragraphs.

In SOE there is not much insight on the resource running behind the service and thus it is hard for users to obtain information that can help them give a correct runtime estimate.

User given estimates are dependent on the user's prior experience with executing similar tasks. Users also tend to overestimate task execution times knowing that schedulers rely on them. In this case a scheduler, depending on the scheduling heuristics, could postpone other tasks due to wrong information. To deal with these scenarios schedulers can implement penalty systems where tasks belonging to these harmful users would be intentionally delayed from execution.

Sometimes it is even difficult for users to provide runtime estimates. These situations usually occur due to the nature of the service. Considering two examples of services, one which processes satellite images and another one which solves symbolic mathematical problems we can draw the following conclusions. In the first case it is quite easy to determine runtime estimates from historical execution times as they depend on the image size and on the required operation. The second case is more complicated as mathematical problems are usually solved by services exposing a Computer Algebra System (CAS). CASs are specific applications which are focused on one or more mathematical fields and which offer several methods for solving the same problem. The choice on which method to choose depends on internal criteria which is unknown to the user. A simple example is given when considering large integer (more than 60 digits) factorizations. These operations have strong implications in the field of cryptography. In this case factorizing n does not depend on similar values as $n\text{-}1$ or $n+1$. Furthermore the factoring time is not linked to the times required to factor $n\text{-}1$ or n. It is therefore difficult for users to estimate runtimes in these situations. Refining as much as possible the notion of similarity between two tasks could be an answer to this problem but in some cases, such as the one previously presented this could require searching for identical past submissions.

Code profiling works well on CPU intensive tasks but fails to cope with data intensive applications where it is hard to predict execution time before all the input data has been received. Statistical estimations of run times face similar problems as code profiling.

Templating has also been used for assigning task estimates by placing newly arrived tasks in already existing categories. General task characteristics such as owner, solver application, machine used for submitting the task, input data size, arguments used, submission time or start time are used for creating a template. Genetic algorithms can then be used to search the global space for similarities.

Despite the difficulty in estimating runtimes there are SAs which do not require them at all. These algorithms take into consideration only resource load and move

tasks only when their loads become unbalanced. This approach works well and tests have shown that scheduling heuristics such as Round-Robin [25] give results comparable to other classic heuristics based on runtime estimates.

In SOE the problem of providing runtime estimates could be overcome by another important aspect related with service costs which is execution deadlines. In this case it does not matter how fast, how slow or where a task gets executed as long as it gets completed inside the specified time interval. Consequently when submitting jobs inside clouds users could attach deadline constraints instead of runtime estimates to either workflows or batch tasks and hope they will not be significantly exceeded. Deadline based scheduling heuristics are specifically useful in cases where users rent services for specific amount of times.

Related with task runtimes is the transfer costs for moving a task from a resource to another. In SOE this is a problem as usually little or nothing is known about the physical location and network route towards a particular service. When moving large amounts of data such as satellite images up to several hundreds of mega-bytes in size the transfer cost becomes an issue. In addition to the time needed to reallocate data problems including licensing and monetary cost arise. There are cases when proprietary data such as satellite images that belong to certain organizations cannot be moved outside their domain (cloud). In this case reallocation to a cloud which provides faster and/or cheaper services for image processing is not possible due to licensing issues.

Task reallocation involves more than simply moving depended data. Clouds rely heavily on virtualization and thus sometimes in orders to execute tasks VMs with certain characteristics need to be created. As a result when reallocating a task the entire VM could require relocation. This implies several other issues such as stopping and resuming preemptive tasks or restarting non-preemptive tasks once they are safely transfered. The problem of transfer costs is thus more problematic than at first glance.

3.2 Service discovery and selection

Services (SOAP-based [36], RESTful [36], Grid Services [20]) are an important part of cloud systems. They allow for software, storage, infrastructure or entire platforms to be exposed through a unitary interface which can be used by third party clients. Each service vendor exposes its services to the general public so that the latter can use them, free or at a cost, in order to solve a particular problem.

Inside this sea of services there is also a constant need of discovering proper services for solving a particular task. Universal Description Discovery and Integration (UDDI) [49] registries offer a solution to this problem. Each service provider registers its services to an UDDI which in turn is used by service consumers for searching specific services. With the occurrence of Web 2.0 these searches could be enhanced with semantic content. Once such a service is found its interface can be

used for submitting tasks and for retrieving their results. Figure 2 shows the typical correspondence between services, UDDIs and clients.

After successfully finding a number of possible candidate services there remains the problem of selecting the best one for the task. In this direction the scheduling heuristics plays an important role as based on several criteria it will select the service which is most likely to minimize the execution costs. It should be noted that depending on whether the scheduling heuristics is adaptive or not a task could be reallocated several times before actually being executed. Task reallocation faces several problems as addressed in Sect. 3.1.

3.3 Negotiation Between Service Providers

Negotiation plays an important role in task scheduling when services from multiple clouds are involved in solving a given problem. Usually the negotiation is linked to the phase of service selection and involves a scheduler request for a particular service characteristic. When considering it smaller execution costs could be achieved.

Negotiation can also involve the decision on what data/tasks are allowed to be submitted to the service and whether the service provider can further use the submitted data/tasks for its own purposes or not.

As most of the times details regarding the VM or application that is exposed as a service are hidden from public the negotiation requires the introduction of negotiator entities which handle pre-selection discussions in the service/cloud name. Usually this stage is accomplished by one or more agents [11], [45]. Details regarding the involved agents will be given in Sect. 5. Depending on the outcome of the negotiation either access to the desired service is either granted or a new negotiation with another agent proceeds.

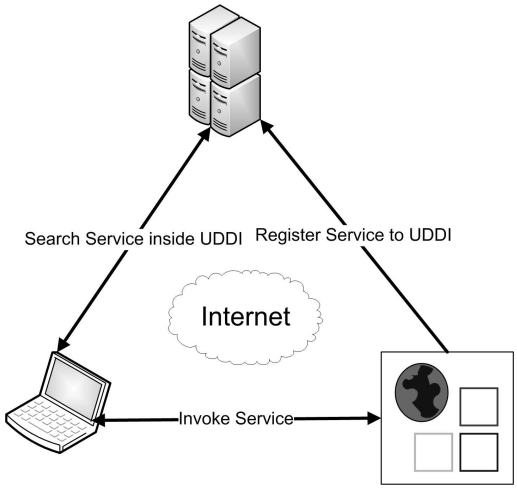

Fig. 2 Finding and invoking services using UDDIs

3.4 Overcoming the Internal Resource Scheduler

An important problem RMSs need to overcome in SOE is that of the internal scheduler used by the service provider. This scheduler is neither influenced nor bypassed by outside intervention. As a result it is said that scheduling between services is accomplished by a meta-scheduler [51] that deals with tasks at service level, leaving the resource level scheduling to the internal Virtual Organization (VO) schedulers (see Fig. 3). These internal schedulers handle tasks assignments depending on their own policies and thus there is no guarantee that the task submitted by the meta-scheduler will be executed inside the cost constraints negotiated at the time of the submission.

As a result of the negotiation between the meta-scheduler and the service provider the latter could try to favor the task by increasing its priority. This action is in the interest of the provider as it could get penalized, with its service trust greatly diminished, for constantly exceeding the imposed deadlines. Consequently future decisions made by the meta-scheduler could ignore the service and the provider would suffer cost losses. We obtain therefore a symbiotic relationship between the meta-scheduler and the service provider that allows both of them to gain advantages: the service provider's trust will increase when executing tasks faster and thus its income will increase by receiving more tasks; and the meta-scheduler will execute tasks faster, minimizing the costs of the client that submitted them.

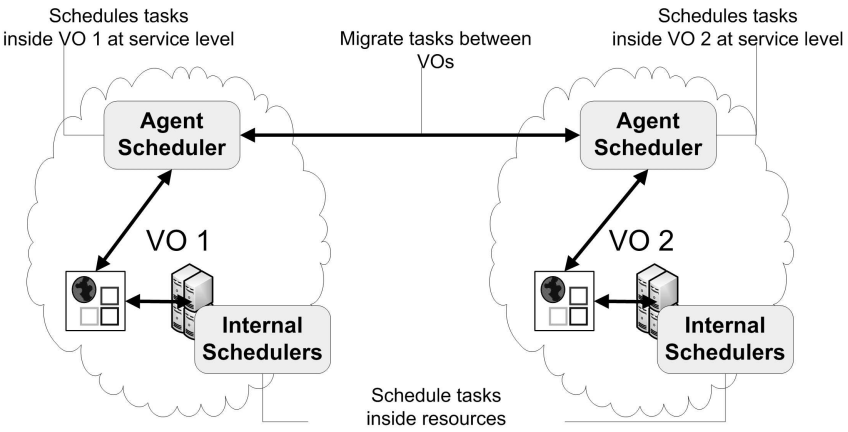

Fig. 3 Scheduling and meta-scheduling in multi-VOs

3.5 Trust in Multi-cloud Environments

When executing tasks on remote services a certain trust level between peers is needed. Trust issued occurs due to many problems including the block box approach of services and because of security issues.

Services cannot be trusted as their interfaces act as black boxes with the content changeable without notice. Thus a service requestor needs to be sure that what it accesses is the same as what was advertised by the service. If this is not the case then the VM running behind the service would not be able to solve the given task inflicting possible cost losses due to time spent for service selection and task submission.

Security issues are also important and are closely linked to the previous problem. These problems can affect both the service requestor and the service provider. The former is usually affected when the data it submits is used for other purposes than those decided during negotiation (e.g. cloning of copyrighted data). The latter can also be affected when data intended to harm the vendor is sent to it. A comprehensive insight on the security issues inside DSs is given in paper [13].

Trust is usually achieved through digital certificates such as the X.509 certificates that are widely used in Web browsers, secure email services, and electronic payment systems.

When using certificates clients usually request one from service providers in order to be granted access.

Web-SOAP and Grid-SOAP services handle security issues by using the WS-Security standard [55]. It allows parties to add signatures and encryption headers to SOAP messages. An extension to WS-Security, WS-Trust [55], deals with issuing, renewing and validating security tokens or broker trust relationships between participants.

In addition to the WS-Security standard the Transport Layer Security (TLS) can also be used. HTTPS for example can be used to cover Web-SOAP, Grid-SOAP and RESTful services.

4 Workflow Scheduling

Workflows consist of several tasks bound together by data/functional dependencies that need to be executed in a specific order for achieving the goal of the problem. They are used especially in cases where the problem can be divided into smaller steps each of them being executed by a distinct solver, or in our case WS. In a cloud environment users usually submit their workflows to a service which orchestrates the execution and returns the result. Whatever happens beyond the service interface is out of reach and invisible to the client. The workflows can be created either by using graphical tools [52] or by directly writing the code in a supported format such as BPEL [53], YAWL [1], Scufl [28], etc. Once the workflow is submitted an enactment engine is responsible for executing the tasks by sending them to corresponding WSs.

In our case these WSs are replaced by scheduling agents that try to schedule the tasks on the best available service through negotiation with other agents. Once a task is completed its result is sent back to the enactment engine which can proceed to the next task and so forth.

An important problem in this communication chain is the return of the result to the workflow engine. To solve this problem the address of the agent responsible for the VO in which the engine is located in is attached to each submitted task. In this way once the execution is completed the result is sent straight back to the agent that initially received the task. This task is usually achieved by messages and will be detailed in Sect. 5.2.

It can be noticed that no prior scheduling decisions are made, and that tasks are scheduled one by one as they become ready for scheduling. This is necessary due to the dynamism and unpredictability of the environment. In paper [24] a unified scheduling model for independent and dependent tasks has been discussed. The goal was to allow SA for independent tasks to be applied to workflows when dynamic environments and online scheduling were considered.

Although this approach is suited when global scheduling decisions are needed there are cases where the workflow engine cannot easily achieve task-to-resource mappings [3] during runtime. Instead workflow SAs such as HEFT [43], Hybrid [44] or CPA [39] could be used. However they only consider the tasks in the current workflow when scheduling or rescheduling decisions are needed. These algorithms provide strategies to schedule workflow tasks on heterogeneous resources based on the analysis of the entire task graph. Every time a workflow is submitted tasks would first be assigned to resources and only then would the workflow execution begin. The negotiation for resources thus takes place prior to runtime. This static approach however is not suited for highly dynamic environments (for example clouds) where: resource availability cannot be predicted; reservations are difficult to achieve; a global perspective needs to be obtained; and deadline constraints require permanent rescheduling negotiations.

In what follows we present an agent-based solution for scheduling workflows. So called scheduling agents are used to negotiate, to schedule tasks and to send the answer back to the workflow engine. Its aim is to provide a platform for online workflow scheduling where tasks get scheduled only when they become ready for execution. This means that a task whose predecessors have not completed their execution is not considered to be submitted for execution.

5 Distributed Agent Based Scheduling Platform Inside Clouds

As clouds are unpredictable in what concerns resource and network load, systems need to be able to adapt to the new execution configurations so that the cost overheads are not greatly exceeded. Multi-Agent Systems (MAS) provide an answer for this problem as they rely on (semi)decentralized environments made up of several specialized agents working together towards achieving a goal through negotiation.

While negotiating each agent keeps a self-centered point of view by trying to minimize its costs.

Although a good option when highly dynamic DS are involved, distributed approaches involve a great amount of transfer overhead [50] as they require permanent updated from their peers in order to maintain an up to date global view. Contrary, centralized approaches do not require a lot of communication but their efficiency peak is maximized mostly when dealing with DS that maintain a relatively stable configuration.

Decentralized agent based solutions for task scheduling also arise as suited solutions when considering a federation of multiple VOs each having its own resources and implementing its own scheduling policies. Submitting tasks in such an environment requires inter-VO cooperation in order to execute them under restrictions including execution deadlines, workloads, IO dependencies etc.

A computing agent can be defined by flexibility, agility and autonomy and as depicted in [21] can act as the brain for task scheduling inside the multi-cloud infrastructure. Agents allow resources to act as autonomous entities which take decisions on their own based on internal logic. Furthermore an intelligent agent [45] can be seen as an extension to the previously given definition by adding three more characteristics: reactivity (agents react to the environment), pro-activeness (agents take initiatives driven by goals) and social ability (interaction with other agents).

In order to take scheduling decisions agents must meet all the previous requirements. They need to quickly adapt to cloud changes and to communicate with others in order to find a suitable service for tasks that need faster execution. In the context of task scheduling agent adaptiveness includes handling changes in resource workload or availability. In what follows we present a SOE oriented agent based scheduling platform.

5.1 The Scheduling Platform

A distributed agent scheduling platform consists of several agents working together for scheduling tasks. Inside a cloud consisting of several service providers (VOs), agents have the role of negotiating and reaching an agreement between the peers. Based on the meta-scheduling heuristics, the internal scheduler, the knowledge on the services it governs and the tasks' characteristics each agent will try to negotiate the relocation from/towards it of several tasks. In trying to achieve this goal agents will also attempt to minimize a global cost attached to each workflow.

Agent based approaches can allow each cloud provider to maintain its own internal scheduling policies [23]. Furthermore they can also use their own scheduling policies at meta-scheduling level. When deciding on task relocations every agent will follow its own scheduling rules and will try to reach an agreement, through negotiation, with the rest. These aspects allow VOs to maintain autonomy and to continue functioning as independents unit inside the cloud. Autonomy is a manda-

tory requirement as VOs usually represent companies that want to maintain their independence while providing services to the general public.

Every VO willing to expose services will list one or more agents to a Yellow Pages online directory which can be queried by other agents wanting to negotiate for a better resource.

Agents can be designed as modular entities. In this way we can add new functionalities to agents without requiring creating new agent types. This is different from previous works [11], [45] which mostly dealt with hierarchies of agents. By doing this we create a super-agent which tries to ensure that the tasks in its domain get the best resources. In addition the need of having multiple agents working together for handling the same task is eliminated. Examples of such agents include: the execution agent, the scheduling agent, the transfer agent, the interface agent, etc.

In our vision all the previously listed specialized agents become sub-modules inside every agent. Thus each agent will have: a scheduling module, a communication module, a service discovery module and an execution module. The sum of all agents forms the meta-scheduler, which is responsible for the inter-service task allocation. Figure 4 details this modular structure together with the interactions between agents and other cloud components.

In what follows we divide the agents in two categories depending on whether they initiate the request i.e. requestor agents, or they respond to an inquiry i.e. solver agents. This division does not influence the characteristics of the agent and is only intended to depict its role.

The *communication module* handles any type of message exchange with other agents. It also facilitates the dialogue between modules such as between the scheduling module and the service discovery module, or between the scheduling module and the executor module.

The *service discovery module* allows each agent to discover services published on UDDI's located inside its own domain. Typically every resource or provider inside a VO willing to offer some functionality to the general public publishes it as services inside an UDDI. Once a service has been published it can be used by the scheduling agent when reallocating tasks. This module is not used to discover services outside the agent's domain. The reason for this behavior is simple: every service outside its domain is not controlled by the agent and thus not trusted. Trust on services is achieved through negotiation with other agents.

The *execution module* is responsible for invoking the service selected for task execution. Service invocation is usually achieved by creating a client tailored to fit the service interface. The creation has to be done dynamically during runtime as it is not feasible to maintain a list of precompiled clients on disc due to the number and diversity of the existing services. Paper [12] presents an API for accessing both SOAP-based services and Grid Services by dynamically creating clients based on the service WSDL (Web Service Description Language) [56]. It should be noted that the execution module is not responsible for creating any VM required by tasks. It is up to the resources behind the service to initialize any required VMs based on the task description.

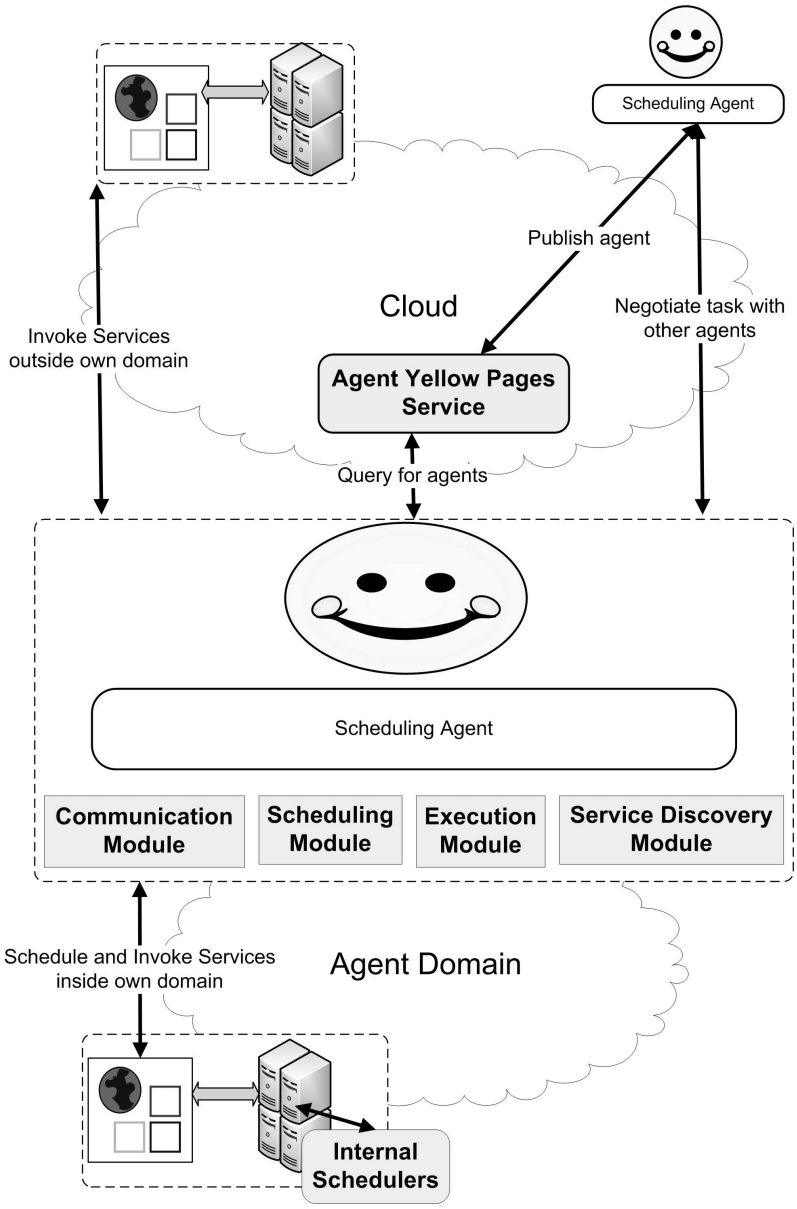

Fig. 4 Agent based scheduling platform

The *scheduling module* deals with task-to-service or task-to-agent allocations. This module is the heart of the agent based scheduling platform and relies on scheduling heuristics for taking its decisions. Every agent has one or more tasks assigned to it. Depending on the scheduling heuristics it can choose to execute some

of the tasks on services governed by agents outside its domain. In the same way it can decide to accept new tasks from other agents.

Depending on the policies implemented by the VO there are two possible scenarios that the scheduling module can face. The first occurs when the agent has no information on the resources running the applications and all it sees are the interfaces of the services. The second is the case when an agent knows all there is to know about the underlying resources i.e. workflow, characteristics, network topology, etc. Both of these scenarios are important depending on how the agent behaves when a task needs to executed on one of its services.

The requestor agent submits the job either directly to the service or to the solver agent. In what concerns the rest of this paper we deal with the latter case. The former option involves bypassing the VO scheduler represented by the agent. This happens because the task will be handled directly by the internal resource scheduler. As a consequence any further scheduling optimization at the meta-scheduling level would be hindered.

The scheduling module inside an agent implements a scheduling heuristics designed to deal with SOE. The scheduling heuristics can be seen as the strategy used by the agent to select a resource for its tasks, while the interaction with other agents represents the negotiation phase. The negotiation proceeds based on rules embedded inside the strategy. A common bargaining language and a set of predefined participation rules are used to ensure a successful negotiation.

Each agent has several services it governs (see Fig. 5). Attached to them there are task queues. Depending on the scheduling heuristics only a given number of tasks can be submitted (by using the execution module) to a service at any given moment. Once submitted to the service it is the job of the internal scheduler to assign the tasks to the resources. Similarly to the service level queues there could also exist queues attached to physical resources. Reallocation between these queues is accomplished by the internal scheduler and is independent on any meta-scheduling decisions taken by agents. Each resource behind a service can implement its own scheduling policies. Usually tasks submitted to a service are not sent back to the agent for meta-scheduling. There are many ways of checking whether a task has been completed or not. One of them requires the scheduling agent to periodically query the service to which it has submitted it for the result. In Sect. 5.3 we briefly present a prototype where internal schedulers have their own scheduling heuristics and work independently from the agent meta-scheduling heuristics.

5.2 Scheduling Through Negotiation

The central entity of every agent based scheduling platform is the scheduling mechanism. Based on its rules agents make active/passive decisions on whether to move or to accept new tasks. Every decision is proceeded by a negotiation phase where the agent requests/receives information from other agents and decides, based on the scheduling heuristics, which offer to accept. Negotiation requires both a language

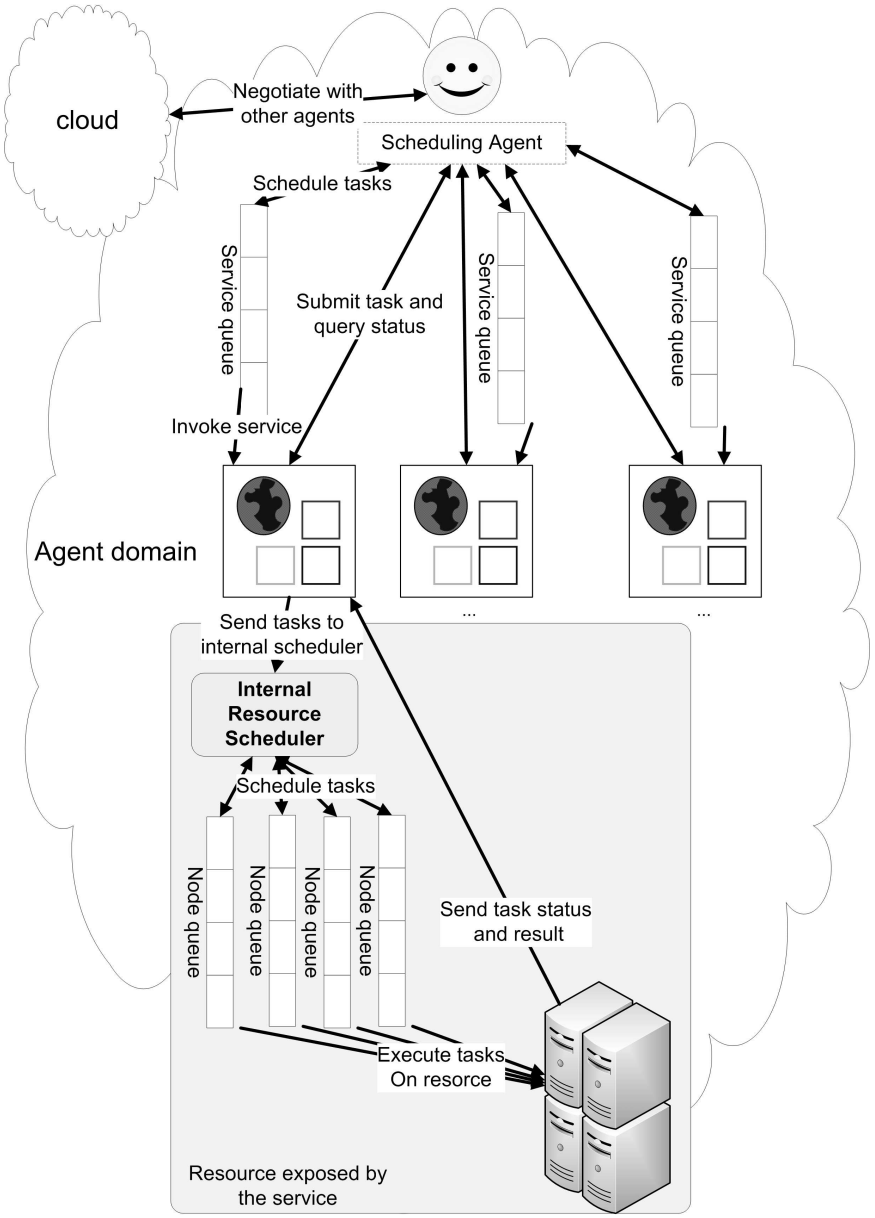

Fig. 5 Task scheduling inside the agent's domain

and a set of participation rules [40]. Depending on the VO policy and on the adherence of other VOs to it many types of negotiation can be used. Examples include game theory models [42], heuristic approaches [19] and argument based [35] solutions.

A minimal set of locutions has been devised for the communication language used by our platform:

- *requestOffer(i,j,k)*: agent *i* requires an offer from agent *j* for a task *k*. Task *k* contains all the information required to make a scheduling decision. This may include (if available): estimated execution times, estimated transfer costs, execution deadlines, required input, etc.;
- *sendOffer(j,i,k,p)*: agent *j* sends an offer of price *p* to agent *i* for the execution of task *k*. The price *p* represents the cost to execute task *k* on resource *j*. Measuring costs depends on the scheduling heuristics. For example it could represent the estimated time required for executing the task on a service belonging to agent *j*;
- *acceptOffer(i,j,k)*: agent *i* accepts the offer of agent *j* for executing task *k*;
- *sendTask(i,j,k)*: agent *i* sends for execution task *k* to a service provided by agent *j*;
- *rejectOffer(i,j,k)*: agent *i* rejects the offer of agent *j* for executing task *k*;
- *requestTasks(i,j)*: agent *i* informs agent *j* that it is willing to execute more tasks;
- *requireDetails(i,j)*: agent *i* informs agent *j* that it requires more details on the services/resources under the latter's management. More specifically they refer to details (WSDL URL for example) on the service proposed by agent *j*;
- *sendDetails(j,i,d)*: agent *j* sends available details to agent *i*. These details contain only publicly available data as result of internal policies;
- *informTaskStatus(i,j,k,m)*: agent *i* informs by using message *m* agent *j* about the status of a task *k*. For example the message could contain the result of a task execution.

Participation rules are required in order to prohibit agents from saying something they are not allowed to say at a particular moment. Figure 6 shows participation rules between these locutions in the form of a finite state machine:

A negotiation starts either from a request for more tasks from an agent *j* or from a request for offers for a given task which an agent *i* decided to relocate. There is a permanent link between the workflow engine agent and the scheduling agent responsible for the VO in which the engine executes. It is to this agent where tasks are placed first. Once a new task has been sent to this agent it is its responsibility to find and negotiate the execution on a resource which has the highest chance of minimizing the deadline constraint. Workflow engines agents are similar with scheduling agents and can communicate with them. However they cannot schedule tasks on resources. Their only purpose is to provide an interface between the engine and the meta-scheduling platform.

When scheduling workflows an important problem that needs to be integrated inside the negotiation phase occurs. Considering the execution of a task on a service that provides a result which can only be further used on services belonging to the same VO, any other possible solutions outside of that VO would be ignored. It is therefore the job of the requestor agent to negotiate for a solution that maximizes the search set. For that reason a balance between the best time cost at a given moment and future restrictions needs to be achieved. As an example, selecting the fastest

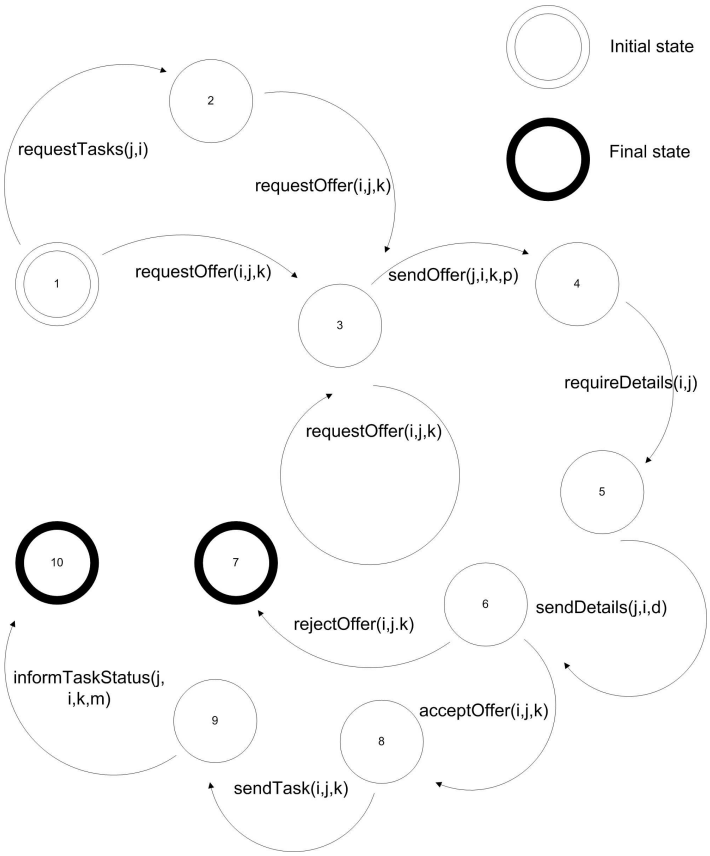

Fig. 6 State transitions between the communication language locutions

service for executing the task could be transformed into selecting the service which executes the task faster and without restrictions on using the result.

In case an agent *j* has requested more tasks from another agent *i* the latter will ask the former for offers regarding the cost of executing some of its tasks. At this point agent *j* will send back to agent *i* an offer for the task in question.

Based on this offer agent *i* will ask for more details regarding the available services which will allow it to make a proper decision: it will either reject or accept the offer. In case agent *i* accepts the offer of agent *j* the task will be submitted to a service queue governed by the latter agent (see Fig. 5). In return it will send back a message on the task status. Once the task is completed the result will be sent back to the workflow agent which will communicate it to the engine. The engine will use the result to select consequent tasks for scheduling and execution.

In the frame of the presented negotiation protocol the key element is played by the moment a request for a relocation offer or for new tasks is made. This point in time basically marks the starting of the negotiation.

The problem of properly selecting the moment of an offer request has been addressed in our paper [22]. The proposed scheduling heuristics incorporates this reallocation moment and it is shown that the schedule outcome is directly influenced by it.

In order to extend this approach to SOE, a deadline-based approach has been investigated in paper [23]. The study is based on the fact that in SOE users usually want to minimize their costs with regard to usage time and thus provide an execution deadline for each task inside their workflows. The aim of the schedule is to minimize the global task lateness i.e. the difference between the actual task finish time and the user given deadline time.

The scheduling heuristics is called DMECT [22] (Dynamic Minimization of Estimated Completion Time). It periodically computes, for every task, the Time Until Deadline (TUD), the Local Waiting Time (LWT) - the time since it was assigned to the current service queue - and the Total Waiting Time (TWT) - time since the task's submission. From these values a decision on whether to move the task or not is taken by checking if the TUD / TWT - LWT is smaller than 0 or not. If the value is smaller a *requestOffer* action is taken. It must be noted that when the decission to relocate the task is taken, all the available services are taken into consideration. These include both internal (part of the current agent domain) and external (obtained from the *requestOffer* inquiry) ones. In this way every existing service gets a fair chance for competing for tasks. It can be easily seen that the relocation relation will try to relocate tasks faster as their deadline approaches.

As a response to a *requestOffer* inquiry, every agent will perform a *sendOffer* action which will inform the requestor agent on possible choices. Every reply typically contains a cost for the task's execution on its best service. If the initial inquiry also contained a lower bound for that cost a list of services offering better prices is returned. The cost for scheduling is made up of execution times possible combined with monetary costs. For example when inquired, each agent will compute the estimated execution time on every service and return only those which have values smaller than the initially provided limit. Alternatively it could return only the smallest value, ensuring that the best available offer it had was made.

In case where it is impossible to estimate the execution time due to insufficient data or internal policies the length of a service queue could be used as measure. In [23] we have shown that the smaller a queue is the likelihood that it executes tasks faster is.

After gathering all the costs the requestor agent will select the best one according to the scheduling heuristics i.e. smallest execution time in our example. All other offers will be rejected. Once selected the task will be sent to the selected solver agent which will place the task in the service queue and the LWT value for the relocated task will be set to 0. In the scenario that the task will not get executed on the newly elected service as well, i.e. TUD / TWT - LWT < 0, the solver agent will send a *requestOffer* inquiry to other agents, thus becoming the newly requestor agent for that task.

Deciding when to request for new tasks is another important case which triggers the negotiation process. In this case an agent sends a *requestTasks* message to all the

other agents informing them about its willingness to accept tasks. Once this message has been sent agents will begin sending *requestOffers* to it for tasks they wish to reallocate. From this point the negotiation proceeds similarly with the previously discussed case.

Depending on the policy the request for new tasks can be done periodically or when the load of the services under an agent's supervision drops below a certain limit. Depending on the scheduling policy this approach of actively searching new tasks could be inefficient. For example in our scenario using the DMECT heuristics such a request would have no effect until at least one task exceeds its staying limit on a resource queue. Other scheduling heuristics based on simple load balancing techniques such as the one presented in [24] could be more suited for this scenario. In these cases there are no conditions preventing tasks from migrating between agents. Once an agent decides that the load on its services has dropped sufficiently new tasks can be requested.

5.3 Prototype Implementation Details

In this section we present some implementation aspects of the scheduling platform prototype. The platform relies on JADE [6] as an agent platform and on the OSyRIS [34] engine for workflow enactment.

JADE facilitates the development of distributed applications following the agent-oriented paradigm and is in fact a FIPA (Foundation for Intelligent Physical Agents) compliant multi-agent middleware. It is implemented in the Java language and provides an Eclipse plug-in which eases the development process by integrating development, deployment and debugging graphical tools. In addition JADE can be distributed across several resources and its configuration can be controlled through a remote graphical user interface. Agents can migrate among these resources freely at any time. Also JADE provides: a standard architecture for scheduling agent activities; a standard communication protocol by using the Agent Communication Language (ACL); and allows the integration of higher functionality by allowing users to include their own Prolog modules for activity reasoning. Even though the simple model of JADE agents makes the development easier it requires a considerable amount of effort for including intelligence when complex control is required.

Paper [38] presents an extension to JADE where the platform is augmented with two types of agents with the aim of paving the way for a more flexible agent cloud system. The two types of agents are: the *BeanShell* agent responsible for sending and executing behaviors coming from other agents; and the *Drools* agent responsible for receiving and executing rules coming from other agents. Authentication and authorization mechanisms are offered for both types of agents.

OSyRIS is a workflow enactment engine inspired by nature where rules are expressed following the Event Condition Action paradigm: tasks are executed only when some events occur and additional optional conditions are met. In OSyRIS events represent the completion of tasks and conditions are usually placed on the

output values. A single instruction is used all the rest (split, join, parallel, sequence, choice, loop) deriving from it: *LHS -> RHS | condition, salience*, where *LHS* (Left Hand Side) represents the tasks that need to be completed before executing the *RHS* (Right Hand Side) tasks. The engine relies on a chemical metaphor where tasks play the role of molecules and the execution rules are the reactions.

In order to simulate VOs we have used two clusters available at the university. One consisting of 8 Pentium dual-core nodes with 4 GB of RAM each (called VO1) and the other having 42 nodes with 8 cores and 8 GB of RAM each. The latter cluster is divided into 3 blades (called VO2, VO2 and VO3) each with 14 nodes each. To each blade there is attached one scheduling agent which manages the services running on them. A single agent is used for governing the entire VO1. Nodes are paired and each pair is exposed through a service handled by the agent handling the governing VO. The agents are registered to a yellow page repository as depicted in Fig. 4. For inter-agent task scheduling the DMECT heuristics is used. Although different SAs could be used for local resource scheduling we have opted for a single one: the MinQL [24] heuristics.

The scheduling scenario proceeds as follows: once a scheduling agent receives a task, it attaches it to one of its service queues (see Fig. 5). Tasks are received either by negotiating with other agents or directly from a workflow agent. The negotiation protocol is similar with the one in Fig. 6 and uses the DMECT SA's relocation condition [23] as described in Sect. 5.2. Each service can execute at most k instances simultaneously. Variable k is equal to the number of processors inside the node pair. Once sent to a service a task cannot be sent back to the agent unless explicitly specified in the scheduling heuristics. Tasks sent to services are scheduled inside the resource by using the MinQL SA which uses a simple load balancing technique. Scheduling agents periodically query the service for completed tasks. Once one is found the information inside it is used to return the result to the agent responsible for the workflow instance. This passes the information to the engine which in turn passes the consequent set of tasks to the agent for scheduling.

In order to simulate the cloud heterogeneity in terms of capabilities services offer different functionalities. In our case services offer access to both CASs and image processing methods. As each CAS offers different functions for handling mathematical problems so does the service exposing it. The same applies for the image processing services that do not implement all the available methods on every service. An insight on how CASs with different capabilities can be exposed as services is given in [37].

6 Conclusions

In this paper we have presented some issues regarding task scheduling when services from various providers are offered. Problems such as estimating runtimes and transfer costs; service discovery and selection; trust and negotiation between providers for accessing their services; or making the independent resource sched-

uler cooperate with the meta-scheduler, have been discussed. As described much of the existing scheduling platforms are grid oriented and cloud schedulers are only beginning to emerge. As a consequence a MAS approach to the cloud scheduling problem has been introduced. MAS have been chosen since they provide greater flexibility and are distributed by nature. They could also represent a good choice for scheduling scenarios where negotiation between vendors is required. Negotiation is particularly important when dealing with workflows where tasks need to be orchestrated together and executed under strict deadlines in order to minimize user costs. This is due to the fact that vendors have different access and scheduling policies and therefore selecting the best service for executing a task with a provided input becomes more than just a simple reallocation problem. The prototype system uses a single type of agents which combine multiple functionalities. The resulting meta-scheduler maintains the autonomy of each VO inside the cloud.

The presented solution is under current development and future tests using various SAs and platform configurations are planned.

Acknowledgements This research is partially supported by European Union Framework 6 grant RII3-CT-2005-026133 SCIEnce: Symbolic Computing Infrastructure in Europe.

References

1. W. M. P. van der Aalst, and A. H. M. ter Hofstede, ”Yawl: yet another workflow language”, Information Systems”, Vol. 30, No. 4, 2005, pp. 245–275.
2. D. Abramson, R. Buyya, and J. Giddy, ”A Computational Economy for Grid Computing and its Implementation in the NIMROD-G Resource Broker”, Future Generation Computer Systems, Vol. 18, No. 8, 2000, pp. 1061–1074.
3. ActiveBPEL, http://www.activebpel.org/ Cited 7 Jan 2010.
4. A. Ali, J. Bunn, et al, Predicting Resource Requirements of a Job Submission, Proceedings of the Conference on Computing in High Energy and Nuclear Physics, 2004.
5. S. Banerjee, I. Mukherjee, and P. K. Mahanti, ”Cloud Computing Initiative using Modified Ant Colony Framework”, World Academy of Science, Engineering and Technology, Vol.56, 2009, pp. 221–224.
6. F. Bellifemine, A. Poggi, and G. Rimassa, ”Jade: a FIPA2000 Compliant Agent Development Environment”, In Proceedings of the fifth international conference on Autonomous agents, 2001, pp. 216–217.
7. F. Berman, R. Wolski, H. Casanova, W. Cirne, et al, ”Adaptive Computing on the Grid using APPLES”, IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 4, 2003, pp. 369–382.
8. R. Buyya, D. Abramson, and J. Giddy, ”Nimrod/g: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid”, Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region, Vol. 1, 2000, pp. 283–289.
9. J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd, ”Arms: An Agent-based Resource Management System for Grid Computing”, Scientific Programming, Vol. 10, No. 2, 2002, pp. 135–148.
10. J. Cao, D. J. Kerbyso, E. Papaefstathiou, and G. R. Nudd, ”Performance Modelling of Parallel and Distributed Computing using PACE”, Proceedings of 19th IEEE International Performance, Computing and Communication Conference, pp. 485–492, 2000.

11. J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd, ”Grid Load Balancing using Intelligent Agents”, Future Generation Computer Systems, Vol. 21, No. 1, 2005, pp. 135–149.
12. A. Carstea, M. Frincu, G. Macariu, D. Petcu, and K. Hammond, ”Generic Access to Web an Grid-based Symbolic Computing Services”, Proceedings of the 6th International Symposium in Parallel and Distributed Computing, 2007, pp. 143–150.
13. A. Carstea, G. Macariu, M. Frincu, and D. Petcu, ”Secure Orchestration of Symbolic Grid Services”, IeAT Technical Report, No. 08-08, 2008.
14. H. Casanova, and J. Dongarra, ”Applying Netsolve’s Network-enabled Server”, EEE Computational Science and Engineering, Vol. 5, No. 3, 1998, pp. 57–67.
15. H. Casanova, G. Obertelli, F. Berman, and R. Wolski, ”The APPLES Parameter Sweep Template: User-level Middleware for the Grid”, Proceedings of Super Computing SC’00, pp. 75–76, 2000.
16. S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, ”Resource Management in Legion”, Future Generation Computer Systems, Vol. 15, No. 5, 1999, pp. 583–594.
17. Cloud Scheduler, http://cloudscheduler.org/ Cited 7 Jan 2010.
18. L. David, and I. Puaut, ”Static Determination of Probabilistic Execution Times”, Proceedings of the 16th Euromicro Conference on Real-Time Systems, pp. 223–230, 2004.
19. P. Faratin, C. Sierra, N. R. Jennings, ”Using similarity criteria to make issue trade-offs in automated negotiation”, Artificial Intelligence, Vol. 142, No. 2, 2001, pp. 205–237.
20. I. T. Foster, ”Globus Toolkit Version 4: Software for Service-Oriented Systems”, Proceedings of International Conference on Network and Parallel Computing, vol. 3779, 2005, pp. 2–13.
21. I. Foster, N. R. Jennings, and C. Kesselman, ”Brain Meets Brawn: Why Grid and Agents Need Each Other”, In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004, pp. 8–15.
22. M. Frincu, ”Dynamic Scheduling Algorithm for Heterogeneous Environments with Regular Task Input from Multiple Requests”, In Lecture Notes in Computer Science, Vol. 5529, 2009, pp. 199–210.
23. M. Frincu, ”Distributed Scheduling Policy in Service Oriented Environments”, Proceedings of the 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2009.
24. M. Frincu, G. Macariu, and A. Carstea, ”Dynamic and Adaptive Workflow Execution Platform for Symbolic Computations”, Pollack Periodica, Akademiai Kiado, Vol. 4, No. 1, 2009, pp. 145–156.
25. N. Fujimoto,, and K. Hagihara, ”A Comparison Among Grid Scheduling Algorithms for Independent Coarse-grained Tasks”, International Symposium on Applications and the Internet Workshops, 2004.
26. S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya, ”Energy-Efficient Scheduling of HPC Applications in Cloud Computing Environments”, Technical Report, 2009.
27. Google Application Engine, http://appengine.google.com Cited 7 Jan 2010.
28. M. Greenwood, ”Xscufl language reference” , 2004, http://www.mygrid.org.uk/wiki/Mygrid /WorkFlow#XScufl workflow definitions Cited 7 Jan 2010
29. B. Lawson, and E. Smirni, ”Multiple-queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems”, In Lecture Notes in Computer Science, vol. 2862, 2002, pp. 72–87.
30. S. Lorpunmanee, M. N. Sap, A. H. Abdullah, and C. Chompooinwai, ”An ant Colony Optimization for Dynamic Job Scheduling in Grid Environment”, World Academy of Science, Engineering and Technology, Vol. 23, 2007, pp. 314–321.
31. M. Maheswaran, T. D. Braun, and H. J. Siegel, ”Heterogeneous Distributed Computing”, Encyclopedia of Electrical and Electronics Engineering, John Wiley & Sons, Vol. 8, 1999, pp. 679–690.
32. Microsoft Live Mesh, http://www.mesh.com Cited 7 Jan 2010.
33. V. Muniyappa, ”Inference of Task Execution Times Using Linear Regression Techniques”, Masters Thesis, Texas Tech University, 2002.
34. OSyRIS Workflow Engine, http://gisheo.info.uvt.ro/trac/wiki/Workflow Cited 7 Jan 2010.

35. S. Parsons, C. Sierra, and N. R. Jennings, ”Agents that Reason and Negotiate by Arguing”, Journal of Logic and Computation, Vol. 8, No. 3, 1998, pp. 261–292.
36. C. Pautasso, O. Zimmermann, and F. Leymann, ”RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision”, 17th International World Wide Web Conference, 2008.
37. D. Petcu, A. Carstea, G. Macariu, and M. Frincu, ”Service-oriented Symbolic Computing with SymGrid”, Scalable Computing: Practice and Experience, Vol. 9, No. 2, 2008, pp. 111–124.
38. A. Poggi, M. Tomaiuolo, and P. Turci, ”Extending JADE for Agent Grid Applications”, Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, 2004, pp. 352–357.
39. A. Radulescu, and A. van Gemund, ”A Low-cost Approach Towards Mixed Task and Data Parallel Scheduling”, Proceedings of the International Conference on Parallel Processing, 2001.
40. S. D. Ramchurn, ”Multi-Agent Negotiation using Trust and Persuasion”, PhD Thesis, University of Southampton UK, 2004.
41. G. R. S. Ritchie, and J. Levine: ”A Hybrid Ant Algorithm for Scheduling Independent Jobs in Heterogeneous Computing Environments”, Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group, 2004.
42. J. S. Rosenschein, and G. Zlotkin, ”Roles of Encounter”, MIT Press, 1994.
43. R. Sakellariou, and H. Zhao, ”Experimental Investigation into the Rank function of the Heterogeneous Earliest Finish Time Scheduling Algorithm”, In Lecture Notes in Computer Science, Vol. 2790, 2003, pp. 189–194.
44. R. Sakellariou, and Zhao, ”A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems”, Proceedings of the 18th International Symposium In Parallel and Distributed Processing, 2004.
45. J. Sauer, T. Freese, and T. Teschke, ”Towards Agent-based Multi-site Scheduling”, Proceedings of the ECAI 2000 Workshop on New Results in Planning, Scheduling, and Design, 2000.
46. W. Shen, Y. Li, H. Genniwa, and C. Wang, ”Adaptive Negotiation for Agent-based Grid Computing”, Proceedings of the Agentcities/AAMAS’02, 2002.
47. W. Smith, I. Foster, and V. E. Taylor, ”Predicting Application Run Times with Historical Information”, Journal of Parallel and Distributed Computing, Vol. 64, No. 9, 2004, pp. 1007–1016.
48. D. Thain, T. Tannenbaum, and M. Livny, ”Distributed Computing in Practice: the Condor Experience”, Concurrency and Computation: Practice and Experience, Vol. 17, No. 2–4, 2005, pp. 323-356.
49. UDDI, www.uddi.org/pubs/uddi-tech-wp.pdf Cited 7 Jan 2010.
50. G. Weichhart, M. Affenzeller, A. Reitbauer, and S. Wagner, ”Modelling of an Agent-based Schedule Optimisation System”, Proceedings of the IMS International Forum, pp. 79–87, 2004.
51. J.B Weissman, ”Metascheduling: a Scheduling Model for Metacomputing Systems”, The Seventh International Symposium on High Performance Distributed Computing, pp 348–349, 1998.
52. P. Wolniewicz, N. Meyer, M. Stroinski, M. Stuempert, H. Kornmayer, M. Polak, and H. Gjermundrod, ”Accessing Grid Computing Resources with G-Eclipse Platform”, Computational Methods in Science and Technologie, Vol. 13, No. 2, 2007, pp. 131–141.
53. WS-BPEL 2.0, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf Cited 7 Jan 2010.
54. WS Security, http://www.ibm.com/developerworks/library/specification/ws-secure/ Cited 7 Jan 2010.
55. WS Trust 1.4, http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html Cited 7 Jan 2010.
56. WSDL, http://www.w3.org/TR/wsdl Cited 7 Jan 2010.

Index