Adaptive Resource Allocation in Cloud Computing Based on Agreement Protocols

Florin Pop, Radu-Ioan Tutueanu, Ciprian Barbieru, Mihaela-Andreea Vasile and Joanna Kołodziej

Abstract In the last years there has been considerable interest in using distributed systems, especially Cloud Systems, in any domains. Resource management contributes to ensure quality of services for any type of application, especially when a system involves elements of heterogeneity characterized by a variety of resources that may or may not be coupled with specific platforms or environments. A problem very close to the industry is the capability to allocate resources in an efficient way and estimate costs, especially when switching from one provider to another. In this chapter we present an extended work oriented on agreement–based resource allocation and a scheduling algorithm, aimed to bring an adaptive fault tolerant distributed system. For the agreement protocol we describe and analyze a 3-TIER structure of resources (hosts and virtual machines). Then an adaptive mechanism for agreement establishment is described. The allocation method considers workload distribution, resources heterogeneity, transparency, adaptability and also the ease to extend by combining with other scheduling algorithms.

1 Introduction

Cloud supports many type of applications like emails, video–streaming, online working environments, file sharing (especially photos and videos), government services or socializing online. Even more, its users vary from children taking pictures with their parents' phone and posting them instantly on social apps (Social Cloud) to engineers using software to aid in their designs. This means having to comply with very broad needs and working with big data. That is why data processing in Cloud systems has gained such an importance. So, everything can be done and kept in the

F. Pop (✉) · R.-I. Tutueanu · C. Barbieru · M.-A. Vasile
Faculty of Automatic Control and Computers, Computer Science Department,
University Politehnica of Bucharest, Bucharest, Romania
e-mail: florin.pop@cs.pub.ro

J. Kołodziej
Cracow University of Technology, Krakow, Poland
e-mail: jokolodziej@pk.edu.pl

J. Kołodziej et al. (eds.), *Intelligent Agents in Data-intensive Computing*,
Studies in Big Data 14, DOI 10.1007/978-3-319-23742-8_9

Cloud. Resource allocation techniques used to support tasks scheduling become critical for Service Level Agreement (SLA) assurance. An adaptive algorithm inspired from SLA would bring several benefits to resource allocation and management, more so when the underlying architecture is highly heterogeneous.

Hybrid Clouds enable keeping sensitive data away from prying eyes, by keeping critical operations on a private cloud, but also leveraging the processing power of public clouds when needed. An agreement protocol would highly benefit resource provisioning, making sure the resources are reserved only when needed—thus increasing the cost effectiveness. Multi-Clouds are known for their high heterogeneity, flexibility and fault tolerance. Having such a protocol that links between the different providers, having the ability to provision resources when needed and establish and honor agreements would make resource management more transparent, more adaptive and cost effective. To sum up, the main benefits gained are *flexibility*—if say a cloud provider increases costs or has downtime, *ability to deal with different clouds*, *transparency*—as the agreements describe clearly the terms, and *costs improvement*.

Sometimes resource and service providers use hybrid solution based on special agreements. The white paper [1] states that even though Big Data is relative to the organization we're talking about, we are definitely in the Big Data era and it is increasing fast throughout a vast number of fields: research, financial services, media, healthcare, defence, human resources and marketing. More and more companies are faced with the challenge of storing and managing huge quantities of data (many PetaBytes, and the order is growing). For Big Data processing we need new tasks scheduling and resource management methods [2].

An important research open issue is transforming that data into information, and that is why Cloud Computing is so important right now. A prediction made by the International Data Corporation [3] says that PaaS industry will see an explosion, as industry public platforms will increase tenfold by 2016 from the number at the end of 2012, many in the domains.

This is why there have been so many different approaches, each trying to solve a particular scheduling problem or to be as general as possible, and take into account multiple cluster parameters. By analyzing a variety of solutions we have come up with an algorithm that can be used both as a task scheduler and resource allocator. Also, the above arguments sustain why scheduling has such a big importance in Cloud Computing. With the size of data, processing it takes huge resources and consumed resources always translate into cost. Another problem in the fact that a company will pay different fees on different Cloud platforms and the price can't be guaranteed regardless of the platform used.

In this chapter we extend our work published in [4], which proposed an agreement based algorithm for task scheduling in Cloud environments with fault tolerance support. There, we proposed a 3-Tier agreement protocol that enables the scheduling algorithm to use about all of the systems already developed in data-centres or in inter-Clouds. The algorithm adds advantages like: workload distribution, heterogeneity, transparency, adaptability and also the ease to extend by combining with other algorithms.

The contributions presented in this chapter, as extensions to our previous results, are:

- we present agreement protocols and establishment algorithms for each of the three TIERS of proposed architecture;
- we describe how the agreement protocols are used to allocate different resources for task scheduling process;
- we present a comparison between a 3-TIER algorithm with difference resource allocation ratio and the default CloudSim allocation algorithm.

The chapter is structured as follows. In Sect. 2 we take a more advanced look at current work in Cloud Computing, scheduling and a few algorithms in particular. Section 3 comes to describe our theoretical solution, focusing on the model and the 3-layer architecture. We describe here the agreement establishment algorithms for each of the layers and the way agreements are used to schedule tasks on different nodes. In Sect. 4 we compare our approach to the scheduling algorithms that come by default with the CloudSim simulator; we compare a 2-TIER algorithm with a 3-TIER algorithm and we take a look at different ratios between the number of nodes on each TIER. Section 5 draws final conclusions, suggest points and ways of improvement of the presented algorithm and also suggest various ways of extending the algorithm.

2 Related Work

Access to a shared pool of resources is the basis of cloud computing and those resources, together with the resources providing access to them are composing the hardware of a cloud environment. The papers [5, 6] do an overall analysis on Cloud Computing in terms of its role and utility, with a big focus on commercial cloud computing solutions, taking into account strengths, weaknesses, opportunities and threats. The authors conclude that providers need to focus on horizontal scalability of the virtualized resources, both by software and hardware means.

With cost effectiveness, performance, flexibility and low power consumption in mind, scheduling is a very important part in cloud computing and it has been given the attention it deserves by researchers and the industry [7, 8]. This is why there have been so many different approaches, each trying to solve a particular scheduling problem or to be as general as possible, and take into account multiple cluster parameters. By analyzing a variety of solutions we have come up with an algorithm that can be used both as a task scheduler and resource allocator.

In general terms, scheduling maps requirements on resources taking into account all requirements. We first have to look at the tasks to be scheduled and run. Tasks can be CPU intensive, needing a lot of processing power and needing no or little input, IO intensive, with little processing and dependent on input/output or a mixed version of the two. Tasks can also vary from Sporadic/Aperiodic Tasks to Bag-of-Tasks, they can have ordering requirements in the form of a DAG or they can belong to a Map—Reduce type of process. A good scheduling algorithm must be aware of all

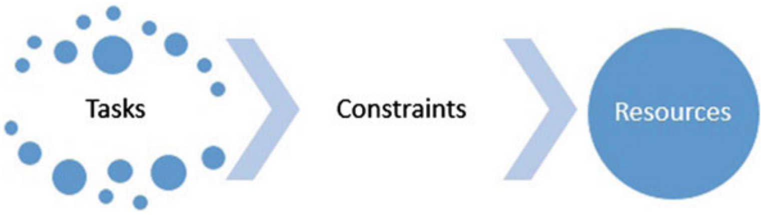

Fig. 1 Scheduling: Tasks mapped on resources, considering the requirements

these requirements and assign tasks accordingly (like the general model presented in Fig. 1).

Another important element in scheduling are resources. They are storage, processing and networking and they can be static or dynamic, provisioned only when needed. Resources are heterogeneous as well; many times a cloud system is made of more than one cluster, each of them with different types of machines and architectures, each of them having its own characteristics. The machines host different virtual machines, each of them with specific tasks. Adding to that, these resources are not static. Machines drop frequently, blades are added, communications could be down. The goal is to have enough resources to satisfy the requests, done through adaptive provisioning, while keeping and energy aware and cost effective system [9].

Regarding resource allocation, a working prototype built on a private Cloud using EUCALYPTUS–based heterogeneous resources was presented by Carrera [10]. The proposed solution monitors the response time of each virtual machine assigned to the farm and adaptively scales up the application to satisfy a SLA promising a specific average response time. Anoter solution that considers SLA for multi-tier applications in Cloud was presented by Iqbal [11]. Here, the agreements are made for automatic detection and resolution of bottlenecks in a multi-tier Web application hosted on a cloud in order to satisfy specific maximum response time requirements. The negotiation oriented on Quality of Service (QoS) is one of the main critical issue for SLA assurance. On the other hand, considering energy–aware allocation of resource, R. Buyya proposed heuristics for provision data center resources to client applications in a way that improves energy efficiency of the data center [12].

Another resource–aware technique that we investigated in our previous work is the dynamic resources allocation model in Grid environments (DyAG) [13]. The proposed solution is responsible with the efficient mapping of the services which make up a Business Process Execution Language workflow onto resources, represented by Web Services, from the Grid environment. This solution is part of DyAG framework (Dynamic resource Allocation in Grid Environments), which is responsible for efficient mapping of abstract invocations with concrete web services which are running on Grid. DyAG scheduler takes into consideration the Opportunistic Load Balancing of the jobs onto the Grid resources. The most important one is the fact that it considers the previous history of the services, both the faults incurred and the jobs which were submitted, and uses this information to predict the future state of the service, thus increasing the chances of avoiding the need for the rescheduling of the jobs which could happen because of various errors. This is an adaptive solution. The DyAG allows the users to dynamically change the policy employed by the scheduler at runtime, through a class loading mechanism. This allows the employment of application profiling techniques in order to finely tune the scheduler in accordance with the characteristics of the environment it is running in, either by changing the various parameters of the policies proposed, or by loading completely new policies.

The main requirement a scheduler needs to be aware of is the deadline. The task has to finish before a specified time and usually this is non-negotiable. Another important requirement is the budget, cost being of great importance in today's industry. Task dependencies need to be satisfied as well, depending of the type of task (DAG, Map—Reduce). Last, but not least, data needs to be transferred between tasks with dependencies, so bandwidth takes an important part in some cases of task scheduling [14–16]. Modified Critical Path is a scheduling algorithm for tasks with dependencies, using Latest Possible Start Time to map tasks to resources [17]. Here, the agreement is made between processors that allow the execution for tasks with the earliest start time. The paper [18] presents a heuristics based genetic algorithm that tries to solve the NP–complete problem of mapping meta-tasks to machine using heuristics like giving priority to the task that can be completed the earliest, combined with a well defined genetic algorithm [19]. The algorithm manages to minimize the completion time and increase the throughput of the system.

Some algorithms have been inspired by P2P networks and distributed hash table implementations like Chord [20], thus having multiple advantages like scalability, decentralization and robustness. In this instance Chord is used to store idle node information in a distributed way. To ensure that QoS metrics are respected, the tasks are split in two categories, one that utilizes a lot of cluster resources but does not run for a prolonged period of time—like Map—Reduce jobs, and another which does not fully utilize the reserved resources but does run for a long time—a web service.

Genetic algorithms, a new evolutionary approach, have been used for resource allocation and task scheduling to increase user satisfaction that is linked to the SLA, and also cloud provider profits [19, 21–23]. A specific example of this approach is presented in [24] where the authors categorize incoming jobs by user priority, which is linked to deadline and cost. Assuming that tasks with a higher priority and shorter SLAs will increase the provider's profit, they designed the fitness function to cater for both user satisfaction and increased income. Another example of bio-inspired applications in this domain is presented in [25], which proposes a scheduling algorithm based on reinforcement learning. This method aims at finding an optimal configuration setting for VMs and software running in those VMs and the experimental results prove the approach's effectiveness.

Another approach is self-adaptive distributed scheduling platform composed of multiple agents implemented as intelligent feedback control loops to support policy–based scheduling and expose self-healing capabilities [26, 27]. While analyzing all of these approaches in parallel, we also tried to compare them so we could find the best solution, like the authors from [28]. They compared a number of 5 algorithms, in terms of complexity and scalability, and also gave recommendations on the environment to use them in (Cloud of Grid). The algorithms explore different directions that vary from SLA trees or MIN–MAX to multi-objective fitness functions.

3 Agreement Protocols for Adaptive Resource Allocation

We apply an agreement based algorithm as mechanism for resource allocation. As tasks scheduling relies on the allocation phase, the agreements represent a base for the scheduling itself. The agreement based approach has many advantages as evenly load resources, add an abstraction layer and the capacity of the algorithm to change as the agreements also change. This algorithm may be used as a starting point for SLAs integration in the scheduling process.

3.1 Modeling and Formulation

We start by describing formally all the elements involved. These are nodes, an abstraction of a VM capable of running on demand tasks, the task itself, the execution time together with dependencies and requirements.

A **node** is an abstraction of VM capable of running on demand tasks. It can have one or more cores, it belongs to a domain—which can be a zone of the same provider or a different provider. The model is

$$N_i(\alpha_i, \beta_i, c_i, a_i, z_i, r_i, q_i), \tag{1}$$

where:

- α_i—execution factor, relative to a reference processor speed

$$\alpha_i = \frac{ProcSpeed(N_i)}{refSpeed}; \tag{2}$$

- β_i—current usage factor:

$$\beta = [\beta_{core_1}, \beta_{core_2}, \ldots, \beta_{core_n}]; \tag{3}$$

- c_i—cost per execution unit;
- a_i—architecture;
- z_i—zone/domain the host belongs to; r_i—current available resources; and
- q_i—current task queue.

A **task** is a bundle of operations to be executed on a node, which may be of one of the types previously presented: Sporadic Task, Bag-of-Tasks, Map—Reduce and

DAG Task. A **task** is defined as

$$T_i(a_i, p_i, d_i, B_i), \tag{4}$$

where:

- a_i—arrival time;
- p_i—estimated processing time on the reference processor;
- d_i—deadline; and
- B_i—budget needed to execute the task.

The **estimated execution time** of a task on a specific node is the product between the task estimate processing time on the reference processor: $ET(t_i, n_j)$—Estimated execution time of task i on node j:

$$ET(T_i, N_j) = \alpha_j p_i. \tag{5}$$

The **dependencies** are modeled as $e_{ij} = (T_i, T_j)$: T_i finishes before T_j starts running.

3.2 Requirements

We have identified the following requirements related to the duration of the agreement, deadlines and the estimated execution time. For task T_i running on node N_j we have:

$$a_i + ET(T_i, N_j) \leq d_i. \tag{6}$$

A **strong agreement requirement** is

$$\max\{a_i + ET(T_i, N_j)\} \leq T_A. \tag{7}$$

A **light agreement requirement** (where $0 \leq f < \frac{1}{n}$ and $n \geq 1$) is

$$\max\{d_i\} \leq T_A + f * T_A. \tag{8}$$

3.3 Architecture Design

We propose a 3-Tier structure considering the following reasons: we have a distributed algorithm and distributing the processing and message passing should show an increase in performance, especially when messages are short; we have machines with different characteristics (OS, architecture) and we could extend the algorithm and specialize those machines with different scopes, for example Tier1 machines specialized on different tasks (see Fig. 2). This would mean we would have a machine

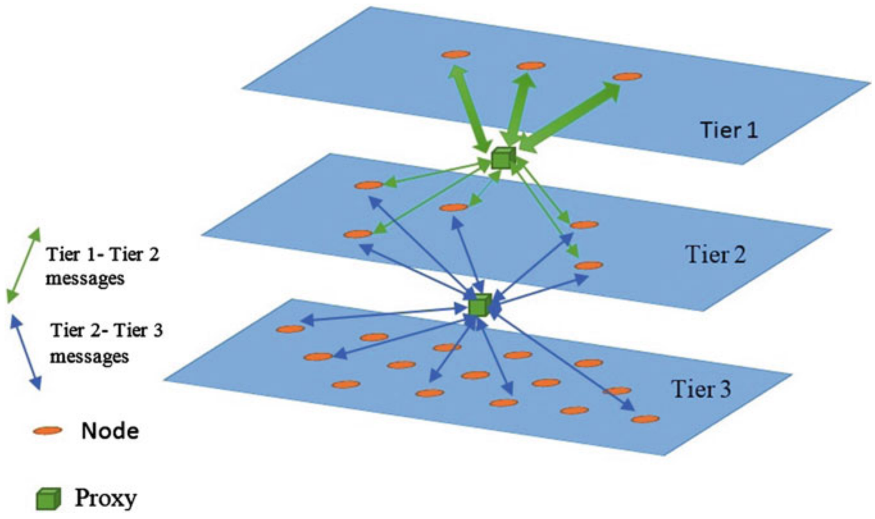

Fig. 2 The *3 layers* and their interaction through proxy servers

specialized in establishing agreements for DAG tasks, another one for sporadic tasks and another for Map—Reduce tasks.

The TIER1–TIER2 separation is inspired from a load balancing system, TIER2 nodes being in charge of distributing tasks to the machines they have an agreement with, distributing the scheduling process and limiting the number of messages that TIER1 computers send. The TIER2–TIER3 separation comes from current solutions in the industry, we have machines that distribute tasks and machines that run them. Here, TIER1 and TIER2 machines distribute tasks, and both TIER3 and TIER2 machines run them. In order to minimize the number of messages used, we added proxy servers between the TIERS. This enables the sender node to send a message containing all the destination nodes, the proxy splitting the message and grouping the resulting messages by destination.

3.4 Agreement and Prerequisites

Agreement. An agreement has two participants: *Initiator* and *Acceptor*. It contains details about the resources reserved and their availability. An agreement is established in two steps for each participant:

```
Initiator:
    1. Request attributes;
    2. Make offer
```

```
Acceptor:
    1. Send attributes;
    2. Accept/Refuse offer.
```

Most importantly, the algorithm is added as a thin layer in middleware, which means that it adds abstraction and it can use all the systems and objects already in place.

We chose an agreement based algorithm due to the many advantages it brings to scheduling in Cloud. One of the first advantages that comes to mind is *workload distribution*, considering load balance and making sure resources on all machines are evenly utilized. It is scientifically proven that machines with a high workload do not function as well and this could lead to unwanted results. Another advantage is *heterogeneity transparency*, the type of VM and the OS being of no importance for the end user. Another feature of the overall system is its *adaptability*, because agreements can change all the time, so when a host is identified as not working properly, agreement establishment with a faulty node can be avoided. This makes the algorithm *fault tolerant*. Its simple design makes it *easy to extend with other algorithms*. We find potential in combining hosts on TIER1 with a genetic algorithm, in order to have specialized TIER1 nodes based on the type of tasks created, the selection of which task is sent to which TIER1 node being determined by a genetic algorithm. They could also learn which machines to establish agreements with, adding a plus of performance to the planning stage. Simplicity also means low complexity, both in understanding and in execution. Last but not least, this algorithm could create a basis for using SLAs. It would add support for the paradigm on lower levels, thus improving both performance and mapping over resources.

Prerequisites. The hosts are referred to as Nodes, each TIER having one or more Nodes. The Nodes know at start up (using a configuration file) which TIER they belong to. In terms of agreement participants, TIER1 nodes are Initiators, TIER3 nodes are Acceptors and TIER2 nodes handle both roles sequentially. They are first Initiators, and after completing agreements with TIER3 nodes, they become acceptors for TIER1 nodes. TIER1 nodes have a mission to fulfill: they need to complete agreements to cover the list of resources needed. TIER3 nodes complete agreements on a first come, first serve basis, while TIER2 nodes need to match resources with requirements (see Fig. 3). The following prerequisites are considered already established:

I. each node has a list with all the other nodes and is aware of which TIER they belong to;
II. A 3-TIER structure has been previously created;
III. TIER1 nodes have a list of resources they need to provide, derived from the tasks that need to be run.

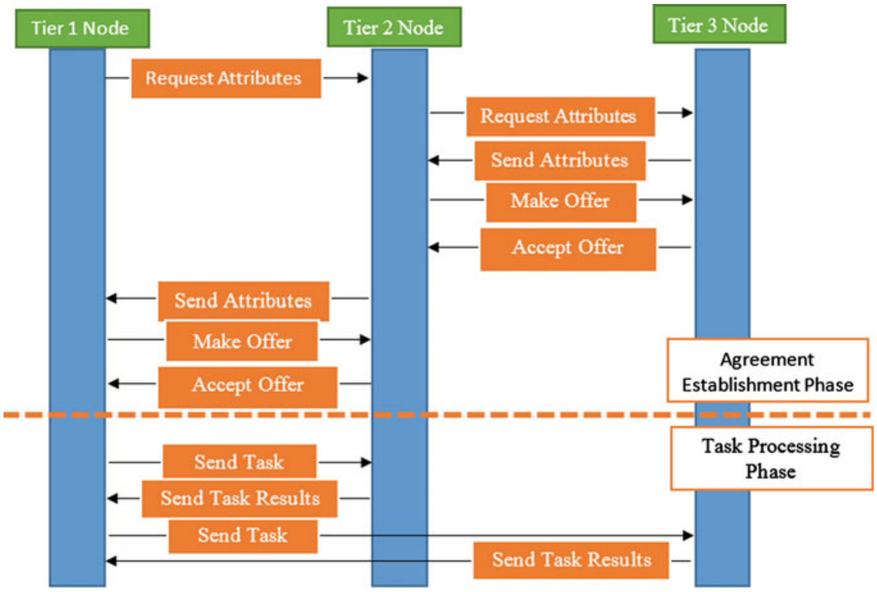

Fig. 3 Overall view of communication between a node on each TIER in the proposed algorithm. The *two phases* are highlighted: agreement establishment and task processing. Here, both nodes accept the offer and are then sent task(s) according to that offer

3.5 Agreement Protocols

TIER1 nodes first ask TIER2 nodes for a list of their attributes (processing power, memory, storage, VM type). Because our model assumes that TIER1 nodes have a list with the tasks that need to be run beforehand, TIER1 nodes have a way to estimate the resources needed (Fig. 4).

First, it sends a request to all nodes from TIER2. They respond to this request with their available resources and resources of nodes they have completed an agreement with, named in the algorithm attributes. Then, they check their estimation of resources needed and make an offer to the respective node, using the smallest number between that of resources needed and that of resources available of that node. After all agreements are established, tasks are assigned to nodes, as described in Fig. 8. Failed tasks are logged, machines are flagged as faulty and the tasks resent to other machines. The list of faulty machines is sent to all other nodes, to prevent further agreements to be established. This uses a 3 strike algorithm, as in an algorithm is considered faulty after three errors.

If the offer is accepted, then the lists that hold info about resources needed (`myResourcesToProvideList`) and resources available (`myResourcesList`) are updated.

```
Algorithm 1 TIER1 AGREEMENT ESTABLISHMENT
for each none $N$ in myNodesList where TIER($N$) = 2 do
  Request Attributes ($N$);
  Add attributes (Node, Resource, Quantity) to AttList;
end for
for each set of attributes ($N$, $R$, $Q$) in AttList do
  Get (from myResourcesToProvideList) the needed resources $\rightarrow$ (myR, myQ);
  (Resource, Quantity) = Compare($R$, $Q$, myR, myQ)
  Make Offer (Node, Resource, Quantity)
  if Offer is Accepted then
    Withdraw(Resource, Quantity) from myResourcesToProvideList;
    Add($N$, Resource, Quantity) to myResourcesList;
  end if
end for
SendTasks(); {tasks are assigned to each node (see Algorithm 5)}
for each Task $T$ failed from node $N$ do
  Add $N$ to faultList; Resend $T$;
end for
for each node $N$ in myNodesList do
  SendFaultlist($N$);
end for
```

Fig. 4 Tier1 Agreement establishment

TIER2 nodes establish agreements with TIER3 nodes similar with the Fig. 4. They send a request for their available resources, attributes in the algorithm, and then they make offers to them. TIER2 nodes have a constant defined, MAX, that represents the maximum number of TIER3 nodes they can establish an agreement with. When an agreement is established, they add the respective node's resources to their list. Fig. 5 describes the way TIER3 nodes establish agreement. They send a list with their available resources when receiving a request, and they accept offers on a FCFS basis. Establishing TIER1–TIER2 agreements from a TIER2 node's perspective are made as follow: TIER2 nodes receive requests for info about their resources and add those requests on a waiting list, until they have established all TIER3 agreements. Only after that they answer to those requests. When receiving an offer, they accept it on a FCFS basis. The agreements are established on a periodic basis, triggered by the system. An interval T is defined and the application autocratically renews agreement at that time. Renewing doesn't necessary mean keeping the old agreements, the whole algorithm is re-run (Figs. 6, 7 and 8).

The algorithm also offers the opportunity for on-demand agreement establishment, in the case of tasks that take more than estimated or failed tasks that need to be

```
Algorithm 2 TIER3 AGREEMENT ESTABLISHMENT
if AttributeRequest has been received from node $N$ (from TIER2) then
  SendAttributes($N$, myResourceList);
end if
if (Offer $O$ is received) AND ($N \notin$ faultList) then
  AcceptOffer($O$);
  remove($O$.Resource, $O$.Quantity) from myAvailableResourcesList;
end if
```

Fig. 5 Tier3 Agreement establishment

Algorithm 3 TIER2 LOWER AGREEMENT ESTABLISHMENT

```
for each node $N$ in myNodesList where TIER($N$) = 2 do
  Request Attributes ($N$);
  Add attributes (Node, Resource, Quantity) to AttList;
end for
$myNodes = 0$;
for each set of attributes ($N$, $R$, $Q$) in AttList do
  if ($myNodes < MAX$) AND ($N \notin faultList$) then
    Make Offer (Node, Resource, Quantity);
    if Offer is Accepted then
      Add($N$, Resource, Quantity) to myResourcesList;
      $myNodes = myNodes + 1$;
    end if
  end if
end for
```

Fig. 6 Tier2 Lower Agreement establishment

Algorithm 4 TIER2 UPPER AGREEMENT ESTABLISHMENT

```
if AttributeRequest has been received from node $N$ (from TIER1) then
  Add $N$ to WaitingList;
  if All lower agreements have been established then
    for each node $N$ in WaitingList do
      SendAttributes($N$, myResourceList);
      Remove $N$ from WaitingList;
    end for
  end if
end if
if Offer $O$ is received then
  if ($N \notin$ faultList) then
    AcceptOffer($O$);
    remove($O$.Resource, $O$.Quantity) from myAvailableResourcesList;
  end if
end if
```

Fig. 7 Tier2 Upper Agreement establishment

Algorithm 5 AGREEMENT BASED RESOURCE ALLOCATION

```
for each task $T$ do
  $scheduled$ = false;
  for each Agreement $A$ do
    if $A.getAvailableRes() > T.getNecessaryRes()$ then
      $scheduled$ = true;
      sendTask($T$, $A$); {Send task to the node at the other side of the agreement
      and update the agreement's available resources.}
      break;
    end if
  end for
  if $scheduled$ is false then
    $A$ = get a new agreement for $T$; {establishes a new agreement that would fit
    requirements for task $T$.}
    sendTask(T, A); {Send task to the node at the other side of the agreement and
    update the agreement's available resources.}
  end if
end for
```

Fig. 8 Agreement based Resource Allocation

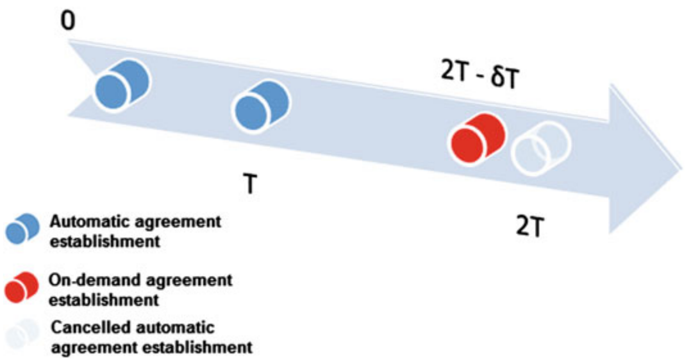
0
2T - δT
T
2T
Automatic agreement establishment
On-demand agreement establishment
Cancelled automatic agreement establishment

Fig. 9 Agreement updates. Skipped automatic agreement example

run on different nodes. The on-demand renewal can cancel the pending future automatic renewal if it is really close to it. Thus, if it is done with less than δT seconds before the pending renewal, where δ is a predefined factor < 1, it delays the renewal, jumping a step for the nodes involved. This can be observed in Fig. 9.

3.6 Adaptive Resource Allocation

The assignment of tasks to resources is done by the TIER1 nodes. To keep the algorithm of low complexity, we schedule the tasks on a FIFO basis. This can be easily changed, wither by using advanced scheduling algorithm at this pace or by making sure the resources are requested in increments that would not make a task have to split to two or more nodes. To avoid adding work for TIER2 nodes without a purpose, TIER1 nodes send tasks directly to the node that runs it (Fig. 8). We cannot talk about an increase in the amount of work TIER1 nodes are doing, because they have to send the same task regardless if they have intermediary nodes or not. Those intermediary nodes would add nothing to the performance of the transport/processing, maybe an extra latency for analyzing the task, deciding that it isn't assigned to them and forwarding it (Fig. 8).

To exemplify that case we will take 3 tasks and we would only consider their processing requirements. Lets say each of the tasks require 5000, 6000 and 7000 million instruction and we have two nodes with a 1000 MIPS processor each. The agreement establishment phase establishes an agreement for 12 s respectively, 6 s. If we execute the 5000 MI task on the second processor, it would remain unoccupied for 1 s and the other 2 processes do no fit in the terms of the other agreement ($6000\,\text{MI} + 7000\,\text{MI} = 130{,}000\,\text{MI} > 1000\,\text{MIPS} * 12\,\text{S}$). However, our algorithm renegotiates agreements every T seconds, where T is very large and also on-demand, so this issue would not be a problem.

3.7 Fault Tolerance

We consider the following faults:

- for *Task* we can have node incompatibility resulting in a processing error, so the task is rescheduled on a different node or even a different platform;
- *Node* N_i is down, so all tasks scheduled on N_i should be rescheduled; and
- $ET(T_i, N_j)$ takes more than initially computed. We can allocate more resources (extend agreement) or, if this is not possible, give task to another node.

In terms of fault tolerance, this agreement based protocol has the potential to use all the systems already in place at the PaaS level. In particular, the algorithm retries a failing task on a different machine. It also logs which machine had failed, creating a faulty machines list. The list is used to avoid agreements with machines, if they have failed more that a predefined number of times, three would be a good number in the general case.

4 Experimental Results

4.1 Methodology and Simulation Scenario

We are also interested in the number of messages, the initial communication not being added to time in our simulation. The delay would be minimum, as we are talking about very short messages, compared to messages used when sending tasks, which usually dwell with large amount of data (even in CPU intensive applications, they would be considerably larger).

We were interested in comparing our algorithm with the base algorithm on time it took for tasks completion and we are expecting similar results. We compared our 3-TIER Agreement Based Algorithm with a 2-TIER Based Algorithm, with the scope of observing the number of messages in between hosts. We are expecting to see an increased number of messages for 3-TIER algorithms. However, the advantage of a 3-TIER algorithm over a 2-TIER algorithm lies not in the total number of messages, but in the fact that they are distributed over the network. We believe that given a very large number of hosts, this would be important and could have performance consequences. We also focus on finding an optimal ratio of hosts on each TIER. We have to keep in mind that agreement establishment is not deterministic, so the results we present are going to be an average of runs. In some cases, we considered better to limit the maximum number of agreements a broker can establish.

Improving Cloud applications is not an easy tasks, testing possible solutions takes a lot of time and resources. Just think you want to test an algorithm that involves 1000 hosts. Those hosts woulds just run the application, and the final product won't be anything useful, your just trying to figure out how good your algorithm these. Because this translates into lots of money spent, simulators have come to solve these problems [29]. CloudSim is a simulation framework at the IaaS level, easily extended, as it it open–source, written in Java. Through cloudlets and VMs, developers can test their algorithms at absolutely no cost.

The simulation layer enables users to model virtual Cloud data–centers, VMs and to manage their memory, storage and bandwidth. This level also enables users to do workload profiling and study their applications' performance [30].

The closest layer to the user of the CloudSim application are the User Interface Structures, which enable users to create and provision VMs and submit cloudlets on those VMs. The VM services handle Cloudlet execution and scheduling and VM management. The Cloud services layer deals with provisioning, be it VM, CPU, Memory, Storage or Bandwidth. On of the lowest layer is that of cloud resources, which deals with event handling, sensors and contains a Cloud Coordinator. The lowest layer is the network layer, which, according to a network topology, adds delayed to messages exchanged between the different entities.

We will now discuss more the main entities used. Hosts control VM operations like: VM provisioning based on a VM allocation policy, VM migration, VM creation and destruction. Hosts represent a physical computer, having a specified processing power(measured in MIPS), memory and storage. These resources have to be shared by the VMs running on that host. The VM allocation policy specifies how VMs are allocated on hosts, whereas the Host allocation policy, taking into account the amount of resources given to each VM, determines how processing power (CPU cores) are assigned to each VM. The Broker then submits Cloudlets to the VM and waits for the results of the execution.

Cost is an important metric in the cloud business, and CloudSim includes this metric via 2 layers. One layer addresses the economics of IaaS, like costs for memory, storage and used bandwidth, while the other one addresses the economics of SaaS, costs for Application Service Requests.

The network is simulated in CloudSim as a message passing interface between DataCenters, DataCenterBrokers and core simulation components. Latencies are added by using a latency matrix that describes the delay of a message traveling from each of the simulation entities to all others. The topology description is stored in the BRITE [31] format, containing a number of network nodes. The CloudSim entities are then mapped to all or part of those nodes, and the topology is used to add delays to the messages.

4.2 Experimental Scenarios

For simulation tests in CloudSim, we used a VM:host ratio of 6:1. We used 4000 independent tasks (simulated as cloudlets) with a number of VMs starting from 120 to 720 using a 100 increment. We used 2 brokers in both cases and we were interested to see the average finishing time. The ratio of TIER2 nodes to TIER3 nodes was 1:6 (see Fig. 10). As the algorithm that we compared (FCFS) our implementation with iterates through VMs and assigns them a Task in order and keeping in mind that our

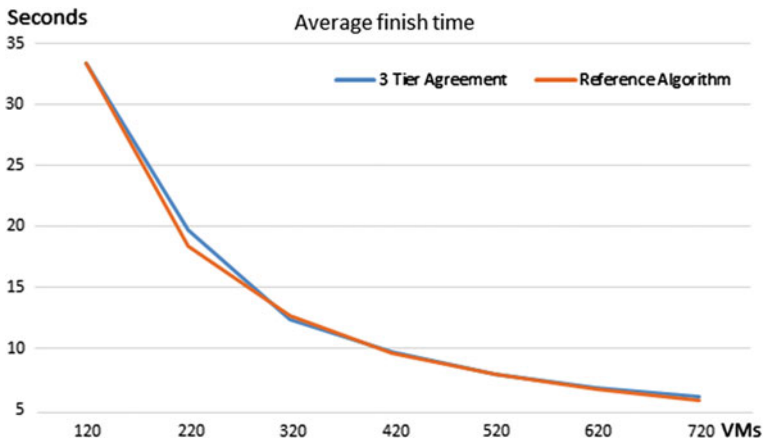

Fig. 10 Comparison between proposed algorithm with a Tier2–Tier3 nodes ratio of 1:6 and the default CloudSim scheduling algorithm

tasks are homogeneous, the fact that the finishing time is so close to it means that we did not loose any machines and comes to validate our algorithm. The agreements are correctly established and tasks are evenly distributed between them.

Hosts have the following characteristics: 2048 MB RAM memory; 1G storage units; 10 k bandwidth unit; equal number of dual-core and quad-core processor, each core with 1000 MIPS while VMs have the following characteristics: image size 10,000 MB; 512 MB RAM; 1 processor, 1000 MIPS; 1000 bandwidth units.

4.3 *Comparative Analysis of Experimental Results*

We used 4000 independent tasks (simulated as cloudlets) with a number of VMs starting from 120 to 720 using a 100 increment. We used 2 brokers in both cases and we were interested to see the average finishing time. The ratio of Tier2 nodes to Tier3 nodes was 1:6 (see Fig. 11). As the algorithm that we compared (FCFS) our implementation with iterates through VMs and assigns them a Task in order and keeping in mind that our tasks our homogenous, the fact that the finishing time is so close to it means that we did not loose any machines and comes to validate our algorithm. The agreements are correctly established and tasks are evenly distributed between them. Figure 12 shows the two slopes one next to the other, better illustrating the similarities between them.

Next, we analyzed two 3-Tier Agreement Based Algorithms, both of them with two Tier1 nodes(brokers), but with different ratios for Tier2 and Tier3. Thus, the first one has three Tier3 nodes for each Tier2 node (a ratio of 1:3), whereas the former has a ratio of 1:6. Figure 11 shows a compared average termination time for 4000 tasks, starting from 120 VMs to 720 using a 100 increment. We can observe

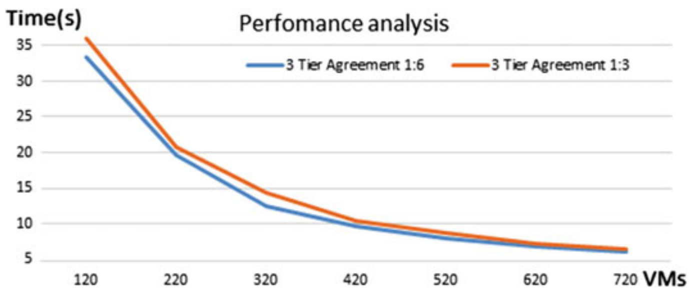

Fig. 11 Comparison between the average termination time of 4000 tasks with different TIER2–TIER3 ratios

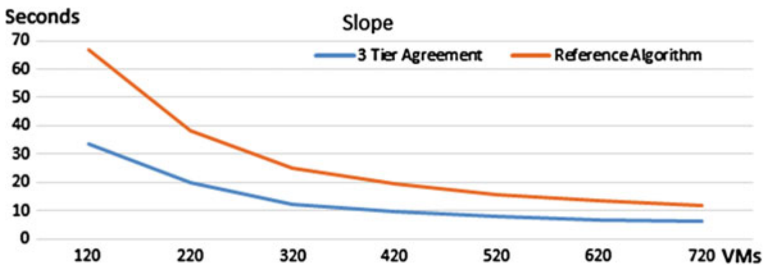

Fig. 12 Comparison between a 3-TIER algorithm with a TIER2–TIER3 nodes ratio of 1:6 (for every 1 VM on TIER2 we have 6 VMs on TIER3) and the default CloudSim scheduling algorithm. The slopes of the two algorithms are very similar, showing almost the same rate of descent

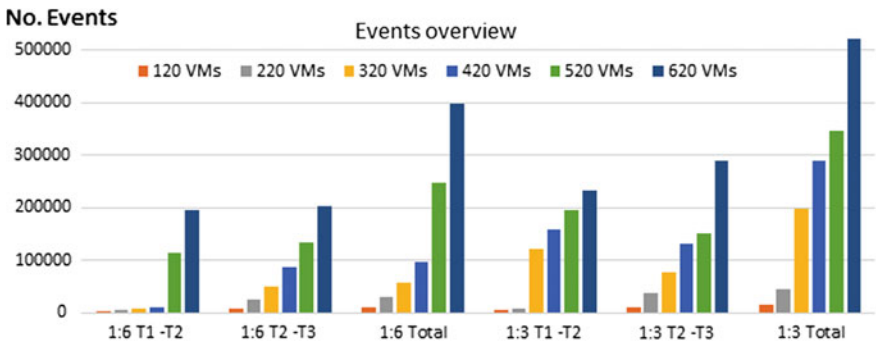

Fig. 13 Comparison between the number of events generated by two 3-TIER Agreement Based Algorithms. We can observe both the total number of events generated, but also a detailed number of events generated for communications between TIER1 and TIER2, TIER2 and TIER3 respectively

that the two approaches offer close result, partly because the tasks are homogeneous, but the algorithm with more nodes on TIER3 performs better.

Figure 13 shows events generated on communications between TIER1 and TIER2, TIER2 and TIER3 respectively. For the 1:6 algorithm, TIER2–TIER3 chatter was

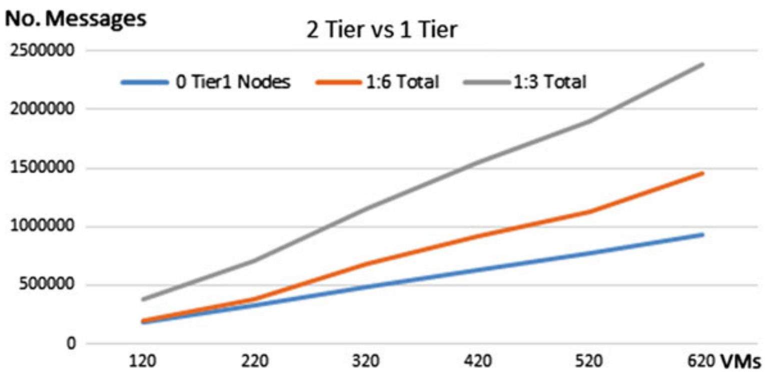

Fig. 14 The number of messages generated by two 3-TIER Agreement Based algorithm and a 2-TIER Agreement Based Algorithm. The first two have a ratio of 1:3 and 1:6 nodes on the second and last TIERS

slightly higher than TIER1–TIER2 and we can notice a big increase with each step in the third cluster, showing the total. The 1:6 algorithm shows the same trend, but it describes an even higher number of events generated.

As a last analysis we take a look at a 2-TIER vs 3-TIER Algorithm comparison. In terms of average finishing times, the first couple of results should apply to this case or, at least, the results should be pretty close, as the default algorithm we compared our implementation to is a 2-TIER algorithm with no agreements. It is important how the number of generated events would evolve for the 2-TIER algorithm, so we added it to the event data presented above. We can observe in Fig. 14 that the number of events generated by a 2-TIER algorithm is, as we would aspect, the lowest, close to that of the 1:3 ratio 3-TIER algorithm.

5 Conclusion

We proposed an agreement based algorithm to enhance the task scheduling process and make it more flexible, transparent and extendible. The algorithm is adaptive, fault tolerant, with workload distribution and large extension capabilities. We tested the proposed algorithm using CloudSim with different numbers of nodes on each TIER. Evidence showed that it is better to have more nodes on the last TIER than on the second, with the 6:1 ratio giving the best results. For future work we propose: defining cost awareness policies for the agreement and extending the (re)negotiation protocol based on that; combining TIER1 nodes with a evolutionary algorithm in order to specialize TIER1 nodes on different types of tasks, specialization that would be seen both in the nodes they establish agreement with and in the algorithm they use to schedule task afterward. Also, agreements are used at higher levels; SLAs have been around for a while now and they are based on negotiation, whereas business agreements have been around for even more time.

We integrated the proposed model into a 3-TIER Agreement Based Algorithm, with nodes on the first TIER establishing agreements with the ones on the second and those on the second TIER establishing agreements with the ones on the third. We have also developed an algorithm that would schedule tasks according to the established algorithms. We have tested the implementation and validated results using CloudSim and its own task scheduling algorithm. We also tested Agreement Based Algorithms with different numbers of nodes on each TIER. Evidence showed that it is better to have more nodes on the last TIER than on the second, with the 1:6 ratio giving the best results.

As the agreement based scheduling is inspired from business agreements, which have been the successful way business is done for hundreds of years, and based on the result obtained, we can assert that this type of scheduling has an important role and can bring important benefits for cloud computing. Therefore, we have identified a few areas which we suggest as future work: defining cost awareness policies for the agreement and extending the (re)negotiation protocol based on that; combining TIER1 nodes with a genetic algorithm or a neural network in order to specialize TIER1 nodes on different types of tasks, specialization that would be seen both in the nodes they establish agreement with and in the algorithm they use to schedule task afterward. In other words, depending on the type of tasks and volume, TIER1 nodes would learn which nodes to use and how to use them; integration with one of the open–source cloud platforms. This would give other users a chance to extend the algorithm and to experience its effects first hand, with an important effect over development of the algorithm.

Acknowledgments The research presented in this paper is supported by projects: *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012; *MobiWay*: Mobility Beyond Individualism: an Integrated Platform for Intelligent Transportation Systems of Tomorrow-PN-II-PT-PCCA-2013-4-0321; *clueFarm*: Information system based on cloud services accessible through mobile devices, to increase product quality and business development farms-PN-II-PT-PCCA-2013-4-0870. We would like to thank the reviewers for their time and expertise, constructive comments and valuable insight.

References

1. Davies, K.: Best practices in big data storage. Tabor Communications Custom Publishing Group (2013). Accessed 20 May 2013
2. Sfrent, A., Pop, F.: Asymptotic scheduling for many task computing in big data platforms. Inf. Sci. **319**, 71–91 (2015)
3. Gens, F.: IDC predictions 2013: competing on the 3rd platform. [Int. Data Corporation] (2012). Accessed 25 May 2013
4. Tutueanu, R.I., Pop, F., Vasile, M.A., Cristea, V.: Scheduling algorithm based on agreement protocol for cloud systems. In: Aversa, R., Koodziej, J., Zhang, J., Amato, F., Fortino, G. (eds.) Algorithms and Architectures for Parallel Processing. Lecture Notes in Computer Science, vol. 8286, pp. 94–101. Springer International Publishing (2013)

5. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
6. Mell, P., Grance, T.: The nist definition of cloud computing (draft). NIST special publication **800**, 145 (2011)
7. Frincu, M.E., Craciun, C.: Multi–objective meta–heuristics for scheduling applications with high availability requirements and cost constraints in multi–cloud environments. In: Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing. UCC'11, pp. 267–274. IEEE Computer Society (2011)
8. Wang, L., Chen, D., Ranjan, R., Khan, S.U., Kołodziej, J., Wang, J.: Parallel processing of massive EEG data with mapreduce. In: Proceedings of the 2012 IEEE 18th International Conference on Parallel and Distributed Systems. ICPADS'12, pp. 164–171. IEEE Computer Society, Washington (2012)
9. Csaji, B.C., Monostori, L., Kfiadfiar, B.: Learning and cooperation in a distributed market-based production control system. In: Proceedings of the 5th International Workshop on Emergent Synthesis, pp. 109–116. Citeseer (2004)
10. Iqbal, W., Dailey, M., Carrera, D.: SLA–driven adaptive resource management for web applications on a heterogeneous compute cloud. In: Cloud Computing, pp. 243–253. Springer (2009)
11. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Gen. Comput. Syst. **27**(6), 871–879 (2011)
12. Beloglazov, A., Abawajy, J., Buyya, R.: Energy–aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems **28**(5), 755–768 (2012). Special Section: energy efficiency in large-scale distributed systems
13. Ion, M., Pop, F., Dobre, C., Cristea, V.: Dynamic resources allocation in grid environments. In: Proceedings of the 2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. SYNASC'09, pp. 213–220. IEEE Computer Society, Washington (2009)
14. Moise, D., Moise, E., Pop, F., Cristea, V.: Resource coallocation for scheduling tasks with dependencies, in grid. In: Procedins of the Second International Workshop on High Performance in Grid Middleware (HiPerGRID), pp. 41–48. IEEE Romania, Bucharest (2008). ISSN: 2065–0701
15. Moise, I., Moise, D., Pop, F., Cristea, V.: Advance reservation of resources for task execution in grid environments. In: Proceedins of the Second International Workshop on High Performance in Grid Middleware (HiPerGRID), pp. 57–64. IEEE Romania, Bucharest (2008). Adaptive Res. Alloc. in Cloud Comp. based on Agreement Protocols 21. ISSN: 2065–0701
16. Vizan, S., Stefanescu, R., Pop, F., Cristea, V.: Decentralized meta–scheduler for economy grid environments. In: Proceedings of the Second International Workshop on High Performance in Grid Middleware (HiPerGRID), pp. 111–122. IEEE Romania, Bucharest (2008). ISSN: 2065–0701
17. Hagras, T., Janeek, J.: Static versus dinamic list–scheduling performance comparison. Acta Polytech. **43**(6) (2003)
18. Kaur, K., Chhabra, A., Singh, G.: Heuristics based genetic algorithm for scheduling static tasks in homogeneous parallel system. Int. J. Comput. Sci. Secur. **4**(2), 183–198 (2010)
19. Kołodziej, J., Khan, S.U.: Multi-level hierarchic genetic-based scheduling of independent jobs in dynamic heterogeneous grid environment. Inf. Sci. **214**, 1–19 (2012)
20. Li, B., Song, A.M., Song, J.: A distributed qos–constraint task scheduling scheme in cloud computing environment: model and algorithm. AISS: Adv. Inf. Sci. Serv. Sci. **4**(5), 283–291 (2012)
21. Iordache, G., Boboila, S., Pop, F., Stratan, C., Cristea, V.: A decentralized strategy for genetic scheduling in heterogeneous environments. Int. J. Multi-agent Grid Syst. **3**(4) (2007). ISSN: 1574–1702
22. Iordache, G., Boboila, S., Pop, F., Stratan, C., Cristea, V.: A decentralized strategy for genetic scheduling in heterogeneous environments. In: Proceedings of on the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, vol. 4276, pp. 1234–1251. Springer, Montpellier Oct 29–Nov 3 (2006). ISBN: 978–3–540–48274–1

23. Iordache, G., Boboila, S., Pop, F., Stratan, C., Cristea, V.: Decentralized Grid Scheduling Using Genetic Algorithms, pp. 215–246. Springer (2008)
24. Jang, S.H., Kim, T.Y., Kim, J.K., Lee, J.S.: The study of genetic algorithm- based task scheduling for cloud computing. Int. J. Control Autom. **5**(4), 157–162 (2012)
25. Xu, C.Z., Rao, J., Bu, X.: Url: a unied reinforcement learning approach for autonomic cloud management. J. Parallel Distrib. Comput. **72**(2), 95–105 (2012)
26. Frincu, M.E., Villegas, N.M., Petcu, D., Muller, H.A., Rouvoy, R.: Self–healing distributed scheduling platform. In: Procedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. CCGRID'11, pp. 225–234. IEEE Computer Society, Washington (2011)
27. Wang, L., Khan, S.U., Chen, D., Kołodziej, J., Ranjan, R., Xu, C.Z., Zomaya, A.: Energy-ware parallel task scheduling in a cluster. Future Gener. Comput. Syst. **29**(7), 1661–1670 (2013)
28. Naik, P., Agrawal, S., Murthy, S.: A survey on various task scheduling algorithms toward load balancing in public cloud. Am. J. Appl. Math. **3**(1–2), 14–17 (2015)
29. Goyal, T., Singh, A., Agrawal, A.: Cloudsim: simulator for cloud computing in-frastructure and modeling. Procedia Eng. **38**, 3566–3572 (2012)
30. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Soft.: Pract. Exp. **41**(1), 23–50 (2011)
31. Medina, A., Lakhina, A., Matta, I., Byers, J.: Brite: An approach to universal topology generation. In: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems. MASCOTS'01, pp. 346–352. IEEE Computer Society, Washington (2001)