2011 35th IEEE Annual Computer Software and Applications Conference Workshops

SLA-Aware Application Deployment and Resource Allocation in Clouds

Vincent C. Emeakaroha, Ivona Brandic, Michael Maurer, Ivan Breskovic
Vienna University of Technology, Vienna, Austria
{*vincent, ivona, maurer, breskovic* }*@infosys.tuwien.ac.at*

***Abstract*—Provisioning resources as a service in a scalable on-demand manner is a basic feature in Cloud computing technology. Service provisioning in Clouds is based on Service Level Agreements (SLAs) representing a contract signed between the customer and the service provider stating the terms of the agreement including non-functional requirements of the service specified as Quality of Service (QoS), obligations, and penalties in case of agreement violations. On the one hand SLA violation should be prevented to avoid costly penalties and on the other hand providers have to efficiently utilize resources to minimize cost for the service provisioning. Thus, scheduling strategies considering multiple SLA parameters and efficient allocation of resources are necessary. Recent work considers various strategies with single SLA parameters. However, those approaches are limited to simple workflows and single task applications. Scheduling and deploying service requests considering multiple SLA parameters such as amount of CPU required, network bandwidth, memory and storage are still open research challenges. In this paper, we present a novel scheduling heuristic considering multiple SLA parameters for deploying applications in Clouds. We discuss in details the heuristic design and implementation and finally present detailed evaluations as a proof of concept emphasizing the performance of our approach.**

***Keywords*-Service Level Agreement, Application Deployment, Service Scheduling Strategies, Cloud Resource Utilization, On-Demand Provisioning**

I. Introduction

Service provisioning in Clouds is based on Service Level Agreements (SLAs) stating the terms for the provisioning including non-functional requirements of the service specified as Quality of Service (QoS), obligations of both parties, and penalties in case of agreement violations.

In order to attain the agreed SLA, the providers must be able to schedule resources and deploy the applications complying with SLA objectives and at the same time optimizing the performance of the applications. Currently, there exists a large body of work considering scheduling of applications in Clouds [6], [10], [12]. These approaches are usually tailored toward one single SLA objective such as execution time, cost of execution, etc. Designing a generalized scheduling algorithm for optimal mapping of workload with multiple SLA parameters to resources is found to be NP-hard due to its combinatorial nature [13]. Viable solutions are based on the use of heuristics.

Currently, in our ongoing FoSII (Foundations of Self-governing ICT Infrastructures) project [5], we are developing models and concepts for autonomic resource management and SLA enforcement in Cloud environments. FoSII infrastructure is intended to be capable of managing the whole lifecycle of self-adaptable Cloud services [1]. We have developed LoM2HiS framework [4] for monitoring Cloud resources, mapping the low-level resource metrics like *system uptime, downtime* to high-level SLA parameters like *availability*, and detecting SLA violations. We have also developed knowledge management techniques [9] for proposing reactive actions to prevent SLA violations and for autonomic management of Cloud resources [7]. Our project lacks a scheduler for efficient deployment of applications considering multiple SLA objectives so far.

In this paper, we present a novel scheduling heuristic considering multiple SLA parameter objectives such as amount of required CPU, network bandwidth, and storage for deploying applications in Clouds. The heuristic includes a load-balancing mechanism for efficient distribution of the applications' execution on the Cloud resources. We also present a flexible on-demand resource allocation strategy included in the heuristic for automatically starting new virtual machines (VM) when a non-appropriate VM is available for application deployment. We discuss the concept and detailed design of the heuristic including its implementation and evaluations using the CloudSim simulation tool [2].

The rest of the paper is organized as follows: Section II presents the related work. Section III explains a deployment model and presents the proposed scheduling heuristic. The implementation issues of the heuristic and extensions to CloudSim are discussed in Section IV. Section V covers the evaluation of the scheduling heuristic in a modeled Cloud environment in CloudSim using heterogenous application workloads. In Section VI, we conclude the paper and describe our future work.

II. Related Work

The existing application scheduling strategies in Clouds are based on approaches developed in related areas such as distributed systems and Grids. Scheduling in these areas is mainly tailored towards ensuring single application SLA objectives. In the Cloud environments on the one hand, applications require guaranteeing numerous SLA objectives to achieve their QoS goals and on the other hand, resource utilization is of paramount importance to the Cloud provider.

Cao *et al.* [3] propose an optimized algorithm for task scheduling based on ABC (Activity Based Costing) in

978-0-7695-4459-5/11 $26.00 © 2011 IEEE
DOI 10.1109/COMPSACW.2011.97

Clouds. They investigate the cost of scheduling different applications and the overhead the applications cause in resource allocation. Their approach considers cost as the only SLA objective for scheduling tasks in a Cloud environment. Lee *et al.* [8] discuss service request scheduling in Clouds based on achievable profits. They propose a pricing model using processor sharing for composite services in Clouds. In their work, two algorithms are devised whereby the first explicitly takes into account not only the profit achievable from the current service, but also the profit from other services being processed on the same service instance. The second algorithm attempts to minimize cost of renting resources from other infrastructure vendors. The approach of this work is similar to ours in the sense that it schedules service requests but differs that they only consider profit objectives for the provider and furthermore, they do not consider resource utilization, like in our case.

Pandey *et al.* [10] discuss a particle swarm optimization-based heuristic for scheduling workflow applications in Cloud computing environments. They focus on minimizing the total execution cost of applications on Cloud resources thereby achieving low computational cost and low data transmission cost. They do not consider resource utilization efficiency and moreover, their approach is targeted at workflow applications.

To the best of our knowledge, none of the discussed approaches deal with service request scheduling considering multiple SLA objectives, high resource utilization strategies, and load-balanced provisioning of resources in a Cloud environment.

III. Scheduling Strategy Design Issues

In this section, we discuss the design of our proposed scheduling heuristic. We first present a use-case scenario showing resource provisioning types in Clouds based on which we position our scheduling heuristic.

A. Cloud Resource Provisioning and Application Deployment

There are three well known types of resource provisioning layers [11] in Clouds: i) *Infrastructure as a Service (IaaS)*; ii) *Platform as a Service (PaaS)*; and iii) *Software as a Service (SaaS)*.

The Cloud provisioning and deployment model presented in Figure 1 shows a use-case scenario of combining the three different layers of resource provisioning to host service requests from customers. The customers place their service deployment requests to the service portal (step 1 in Figure 1), which passes the requests to the request processing component to validate the requests (step 2). If the request is validated, it is then forwarded to the scheduler (step 3). The scheduler selects the appropriate VMs through the provisioning engine in *PaaS* layer for deploying the requested service and the load-balancer balances the service

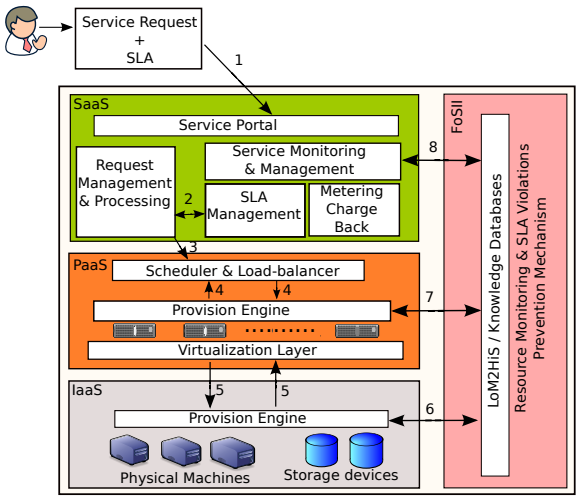

Figure 1. Cloud Provisioning and Deployment Model.

provisioning among the running VMs (step 4). The provision engine manages the VMs on the virtualization layer and the virtualization layer interacts with the physical resources via the provision engine in *IaaS* layer (step 5).

The low-level resource metrics of the physical resources at the *IaaS* layer are monitored by the *LoM2HiS* framework [4] (step 6). The knowledge database techniques [9] in FoSII provide reactive actions in case of SLA violations (step 7). The service status and the SLA information are communicated back to the service portal (step 8).

There is the possibility of provisioning at the single layers alone. However, our approach aims to provide an integrated resource provisioning strategy. Thus, our proposed scheduling heuristics considers the three layers. Efficient resource provisioning and application deployments at these layers are not trivial considering their different constraints and requirements. At the *IaaS* layer the physical resources must be managed to optimize utilizations. At the *PaaS* layer, the VMs have to be deployed and maintained on the physical host considering the agreed SLAs with the customer. Deploying single applications at *SaaS* layer is challenging due to the fact that each application demands the fulfillment of its SLA terms. In the next section we discuss our proposed scheduling heuristic aimed to address these challenges.

B. Scheduling Heuristic Description

The proposed scheduling heuristic aims to schedule applications on VMs based on the agreed SLA terms and deploy VMs on physical resources based on resource availabilities. With this strategy application performance is optimized while the possibilities of SLA violations are reduced. Moreover, the integrated load-balancer in the heuristic ensures high and efficient resource utilization in the Cloud environment.

Algorithm 1 Scheduling Heuristic

```
1: Input: UserServiceRequest
2: get globalResourcesAndAvailableVmList;
3: // find appropriateVMList
4: if AP(R, AR) ! = ∅ then
5:    //call the load balancing algorithm
6:    deployableVm = load-balance(AP(R, AR))
7:    deploy service on deployableVm;
8:    deployed = true;
9: else
10:   if globalResourceAbleToHostExtraVM then
11:      start newVMInstance;
12:      add VMToAvailableVMList;
13:      deploy service on newVm;
14:      deployed = true;
15:   else
16:      queue serviceRequest until
17:      queueTime > waitingTime
18:      deployed = false;
19:   end if
20: end if
21: if deployed then
22:   return successful;
23:   terminate;
24: else
25:   return failure;
26:   terminate;
27: end if
```

As shown in Algorithm 1, our scheduler receives as input the customers' service deployment requests (R) that are composed of the SLA terms (S) and the application data (A) to be provisioned (line 1 in Algorithm 1).

The output of the scheduler is the confirmation of successful deployment or error message in case of failure. In the first step, it extracts the SLA terms, which forms the basis for finding the VM with the appropriate resources for deploying the application. Next, it gathers information about the total available resources (AR) and the number of running VMs in the data center (line 2). The SLA terms are used to find a list of appropriate VMs (AP) capable of provisioning the requested service (R) (lines 3-4).

Once the list of VMs are found, the load-balancer decides on which particular VM to deploy the application in order to balance the load in the data center (lines 5-8).

In case there is no VM with the appropriate resources running in the data center, the scheduler checks if the global resources consisting of physical resources can host new VMs (lines 9-10). If that is the case, it automatically starts new VMs with predefined resource capacities to provision service requests (lines 11-14). When the global resources cannot host extra VMs, the scheduler queues the provisioning of service requests until a VM with appropriate resources is available (lines 15-16). If after a certain period of time, the service requests cannot be scheduled and deployed, the scheduler returns a scheduling failure to the cloud admin, otherwise it returns success (lines 17-27).

Algorithm 2 Load Balancing Strategy

```
1: Input: AP(R, AR)
2: globalvariable availableVmList
3: globalvariable usedVmList;
4: deployableVm = null;
5: if size(usedVMList) == size(availableVmList) then
6:    clear usedVmList;
7: end if
8: for vm in AP(R, AR) do
9:    if vm not in usedVmList then
10:      add vm to usedVmList;
11:      deployableVm = vm;
12:      break;
13:   end if
14: end for
15: return deployableVm;
```

The load-balancer is presented in Algorithm 2. It receives as input the appropriate VM list (line 1 in Algorithm 2). In its operations, it first gets the number of available running VMs in the data center in order to know how to balance the load among them (line 2). In the next step, it gets a list of used VMs, i.e., VMs that are already provisioning applications (line 3). If this list is equal to the number of running VMs, it clears the list because that means all the VMs are currently provisioning some applications (lines 4-7). Therefore, the first VM from the appropriate VM list can be selected for the deployment of the new application request. The selected VM will then be added to the list of used VMs so that the load-balancer will not select it in the next iteration (lines 8-15).

IV. Implementation Issues

The proposed scheduling heuristic is implemented as a new scheduling policy in the CloudSim simulation tool for the purpose of evaluation. CloudSim is a scalable simulation tool offering features like support for modeling service brokers, resource provisioning, application allocation policies, and simulation of large scale Cloud computing environments including data centers, on a single computing machine. Further information about CloudSim can be found in [2].

A. Custom Extensions and Scheduler Implementation

We extended CloudSim with the components shown in the *custom extensions layer* as presented in Figure 2. The infrastructure level services are modeled by the *core layer* representing the original CloudSim data center, which encapsulates sets of computing hosts that can either be homo-

geneous or heterogeneous with respect to the configuration of their hardware (CPU cores, bandwidth, storage, memory).

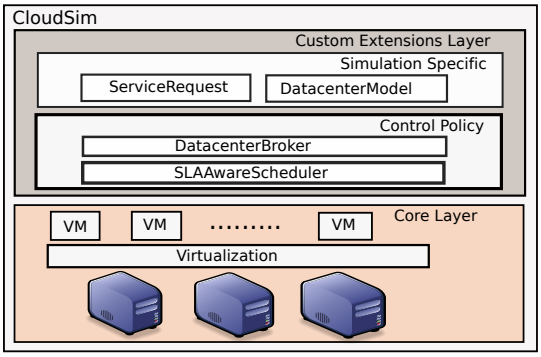

Figure 2. CloudSim Extension Architecture.

Our extensions to CloudSim are divided into two groups of Java classes: i) the *control policy* classes; and ii) the *simulation specific* classes. The *control policy* classes include the implementations of a new *data center broker* for interfacing with the data center and our proposed scheduling heuristic. The *data center broker* is responsible for mediating negotiations between customers and Cloud providers in respect to allocating appropriate resources to customer services to meet their application's QoS needs and to manage the providers resources in the CloudSim. The new *data center broker* includes the capability of running dynamic simulations thereby removing the burden of statically configuring the whole simulation scenario before starting the simulation.

The proposed scheduling heuristic provides policies used by the *data center broker* for allocating VM resources to applications. The implementations of the heuristic and that of the load-balancer are realized with Java methods in a class named *SLAAwareScheduler* as shown in Figure 2. This class is used by the *DatacenterBroker* class to schedule, deploy applications, and manage the data center resources.

The *simulation specific* classes are used in realizing simulation scenarios. This group includes two Java classes named *DatacenterModel* and *ServiceRequest* as shown in Figure 2. The *DatacenterModel* class presents methods for flexible instantiation of different data center scenarios for scalable simulations. We used this class in our evaluations to easily configure and evaluate different scenarios. The *ServiceRequest* class represents a customer service request. It encapsulates information about the SLA parameter objectives and the application data to be provisioned in the Cloud.

V. Evaluation

In this section, we discuss the evaluation of the scheduling heuristic. The evaluation demonstrates the resource utilization achievable by the scheduler. It further shows the higher application performance obtainable while compared to an arbitrary task scheduler. We first describe the experimental setup and configurations.

A. Basic Experimental configurations

Our experimental testbed is setup as described in Figure 1. It demonstrates the processes of placing service request by customers and how our proposed scheduler deploys the service on appropriate Cloud resources.

The Cloud resources comprises physical and virtual machines. Table I shows the resource capacities of the physical machines and the configuration parameters of the virtual machines. Based on the capacities of the physical machine resources and the sizes of the virtual machines, we can start several virtual machines on one physical host.

Table I
Cloud Environment Resource Setup

Machine Type = Physical Machine					
OS	**CPU Core**	**CPU Speed**	**Memory**	**Storage**	**Bandwidth**
Linux	6	6000 MIPS	3,072 GB	30000 GB	3 Gbit/s
Machine Type = Virtual Machine					
OS	**CPU Core**	**CPU Speed**	**Memory**	**Storage**	**Bandwidth**
Linux	1	1000 MIPS	512 MB	5000 GB	500 Mbit/s

In our evaluations, we use two types of applications to realize heterogeneous workloads. The first workload is extracted from a Web Application (WA) for an online shop and the second workload is a trace of High Performance Computing (HPC) application represented by an image rendering applications such as POV-Ray [1].

Table II presents our experimental SLA objective terms for the two application types. The web application generally requires less resources in execution while the HPC applications are resource intensive. The SLA objectives presented in Table II must be guaranteed to ensure the performance of the applications.

Table II
Heterogenous Application SLA Objectives

Application Type	**CPU Power**	**Memory**	**Storage**	**Bandwidth**
Web	240 MIPS	130 MB	1000 GB	150 Mbit/s
HPC	500 MIPS	250 MB	2000 GB	240 Mbit/s

B. Deployment Efficiency and Resource Utilization

In this experiment, we evaluate the efficiency of the proposed scheduler for deploying customer service requests and utilizing the available Cloud resources. Furthermore, we test the essence of the on-demand resource provisioning feature, which dynamically starts new VMs. We create a large data center made up of 60 physical machines and 370 virtual machines. We generate and use 1500 service requests for the experiment.

To evaluate the capabilities of the scheduler, we divide our evaluation into two groups: i) fixed resource and ii) on-demand resource. In the fixed resource group the on-demand resource provisioning feature is deactivated while

[1] www.povray.org

in the on-demand resource group it is activated. The essence of these two groups is to demonstrate the advantages of the on-demand resource provisioning feature. Each group runs three scenarios: i) the first scenario handles the deployment of *only web applications'* service requests; ii) the second scenario deals *only with HPC applications*; and iii) the third scenario deals with a *mixture of web and HPC applications*.

The three scenarios are intended to cover real world deployment situations where applications exhibit different resource consumption behaviors.

Table III
ACHIEVED DEPLOYMENT AND RESOURCE UTILIZATION RESULTS

Fixed Resource Group			On-demand Resource Group	
Scenario	**Utilization**	**Deployment**	**Utilization**	**Deployment**
1	100%	98.67%	100%	100%
2	100%	49.67%	100%	80%
3	94.05%	61.73%	98%	100%

Table III presents the overall achieved results in scheduling and deploying the applications of the three scenarios. In the following we describe the results.

Fixed resource group: In this case the scheduler schedules and deploys the applications on the available running VM in the data center without the flexibility of starting new VMs when required. The results achieved in this group is presented in the left side of Table III.

In the first scenario, the scheduler achieved 100% resource utilization implying full utilizations of VM resources. The resource utilization is measured by checking the number of service applications the scheduler can deploy on each VM in relation to the resource capacity of the virtual machine. The deployment efficiency is calculated by counting the total number of deployed service applications in relation to the total number of requested services. In this scenario a total of 1480 service applications are deployed whereas a total of 1500 service requests were made. This gives a deployment efficiency of 98,67%. About 20 service requests could not be provisioned due to lack of resources on the available VMs.

The scheduler in the second scenario achieved 100% resource utilization in this case. However, it achieved only 49.73% deployment efficiency. The low deployment efficiency is caused by lack of available resources because the HPC applications are resource intensive in computing thereby easily consuming the available resource in the data center.

In the third scenario, approximately an equal number of service requests for both application types are generated. The scheduler achieved about 94.05% resource utilization in this scenario. The inability to achieve 100% resource utilization is caused by the heterogenous nature of the workload whereby some HPC applications cause some resource fragmentation leaving some resource fragments that are not usable by the scheduler. Furthermore, it achieved 61.73% deployment efficiency. This is significantly better than the deployment efficiency achieved in the second scenario. This increase in deployment efficiency is attributed by the heterogenous workload whereby the number of HPC applications' requests is smaller than in the second scenario.

On-demand resource group: In this group, the scheduler can flexibly start new VMs when necessary as far as there are available physical resources. The results obtained by the three evaluation scenarios are depicted in right side of Table III. In the first scenario, the scheduler achieved 100% utilization. The interesting observation in this scenario compared to the first scenario of former group, is the 100% deployment efficiency achieved. The scheduler made advantage of the flexible on-demand resource provisioning feature to start extra four VMs to fully deploy the whole service requests.

The second scenario achieved 100% resource utilization. Although the resources were fully utilized, the scheduler could only achieve 80% deployment efficiency. This is a far better result than 49.33% achieved by the equivalent scenario in the former group. The scheduler created extra 229 VMs for the applications deployments thereby reaching the limits of the physical resources. The scheduler could not achieve 100% deployment efficiency due to an ultimate lack of resources in the data center. This problem is part of our future work where we will consider application scheduling in federated cloud environments.

The third scenario achieved 98% resource utilization due to resource fragmentations caused by the heterogenous workload and resource over-provisioning. The last two VMs started on-demand were under-utilized. A 100% deployment efficiency was achieved in this scenario by starting 215 VMs on-demand.

Comparing the results achieved by the fixed resource group scenarios against those of the on-demand resource group (Table III), it can be clearly seen that the later group obtained much better resource utilization rates and deployment efficiencies. This demonstrates the effectiveness and relevance of our proposed scheduling approach in a Cloud environment.

C. Application Performance Comparison

In this section we discuss the performance of the applications being provisioned in our Cloud testbed. The application performance is evaluated in two aspects using the scenarios of the previous section: i) response time for the web applications and ii) completion time for the HPC applications. We compare the result achieved by our proposed scheduler with that achieved by an arbitrary task scheduler.

Table IV presents the applications performance results. The results show the average response time and completion time of the applications while deployed by the two schedulers. It can be clearly seen that our proposed scheduler is two times better than the task scheduler. The good performance of our scheduler is attributed to the fact that it

Table IV
SCHEDULER COMPARISON

Without On-demand Resource Provisioning Feature

	SLA-aware Scheduler		Traditional Task Scheduler	
Scenario	**Response Time**	**Completion Time**	**Response Time**	**Completion Time**
1	8sec	-	20sec	-
2	-	10sec	-	22sec
3	10sec	14sec	25sec	30sec

With On-demand Resource Provisioning Feature

	SLA-aware Scheduler		Traditional Task Scheduler	
Scenario	**Response Time**	**Completion Time**	**Response Time**	**Completion Time**
1	5sec	-	15sec	-
2	-	7sec	-	18sec
3	8sec	10sec	19sec	24sec

considers multiple performance objectives before deciding on which resource to deploy an application thereby finding the optimal resource for the application best performance, whereas the task scheduler considers mainly single objectives in its deployment, which can not provide the optimal resources for the application best performance. Note that in Table IV the on-demand resource provisioning feature applies only to our proposed scheduler.

VI. CONCLUSION AND FUTURE WORK

Provisioning customer applications in the Cloud while maintaining the application's required quality of service and achieving resource efficiency are still open research challenges in Cloud computing. Scheduling and deployment strategies are means of achieving resource provisioning in Cloud environments.

In this paper, we presented a novel scheduling heuristic considering multiple SLA objectives in deploying applications in Cloud environments. The heuristic includes load-balancing mechanism for efficient distribution of the applications' execution among the Cloud resources. We also presented a flexible on-demand resource usage feature included in the heuristic for automatically starting a new VM when non-appropriate VM is available for application deployment. We discussed in details the design of the heuristic and its implementations.

We evaluated our proposed scheduling heuristic using the CloudSim simulation tool. We used different evaluation scenarios to test the resource utilizations and deployment efficiencies our approach can achieve and to investigate the performance of the applications being provisioned.

In our future work, we will investigate scheduling and application deployment in Cloud considering energy efficiency objectives in allocating and utilizing resources. Furthermore, we will study scheduling and deployment of applications in federated Cloud environments investigating different outsourcing strategies.

ACKNOWLEDGMENT

This work is supported by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII). We would like to thank Rodrigo Calheiros for his support in extending CloudSim.

REFERENCES

[1] I. Brandic. Towards self-manageable cloud services. In *33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09)*, 2009.

[2] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. In *Software: Practice and Experience*. Wiley Press, New York, USA, 2010.

[3] Q. Cao, Z.-B. Wei, and W.-M. Gong. An optimized algorithm for task scheduling based on activity based costing in cloud computing. In *3rd International Conference on Bioinformatics and Biomedical Engineering ICBBE 2009.*, pages 1 –3, June. 2009.

[4] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar. Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In *High Performance Computing and Simulation Conference*, pages 48 – 55, Caen, France, 2010.

[5] FoSII. Foundations of self-governing ict infrastructures. http://www.infosys.tuwien.ac.at/linksites/FOSII/index.html.

[6] S. K. Garg, R. Buyya, and H. J. Siegel. Time and cost trade-off management for scheduling parallel applications on utility grids. *Future Gener. Comput. Syst.*, 26(8):1344–1355, 2010.

[7] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[8] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou. Profit-driven service request scheduling in clouds. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), 2010*, pages 15 –24, may. 2010.

[9] M. Maurer, I. Brandic, V. C. Emeakaroha, and S. Dustdar. Towards knowledge management in self-adaptable clouds. In *4th International Workshop of Software Engineering for Adaptive Service-Oriented Systems (SEASS'10)*, Miami, Florida, USA, 2010.

[10] S. Pandey, L. Wu, S. M. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *AINA '10: Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pages 400–407, Washington, DC, USA, 2010. IEEE Computer Society.

[11] R. Prodan and S. Ostermann. A survey and taxonomy of infrastructure as a service and web hosting cloud providers. In *10th IEEE/ACM International Conference on Grid Computing, 2009*, pages 17 –25, October 2009.

[12] M. Salehi and R. Buyya. Adapting market-oriented scheduling policies for cloud computing. In *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 351–362. Springer Berlin / Heidelberg, 2010.

[13] J. M. Wilson. An algorithm for the generalized assignment problem with special ordered sets. *Journal of Heuristics*, 11(4):337–350, 2005.