Journal of King Saud University – Computer and Information Sciences (2017) **29**, 19–28

King Saud University

**Journal of King Saud University – Computer and Information Sciences**

www.ksu.edu.sa
www.sciencedirect.com

# A novel agent based autonomous and service composition framework for cost optimization of resource provisioning in cloud computing

**Aarti Singh** [a,*], **Dimple Juneja** [b], **Manisha Malhotra** [a]

[a] *MMICT & BM, M.M. University, Haryana 133207, India*
[b] *DIMT, Kurukshetra, Haryana, India*

Received 9 April 2015; revised 3 August 2015; accepted 12 September 2015
Available online 17 November 2015

**KEYWORDS**

Cloud computing;
Cloud mobile agent;
Cost optimization;
Resource allocation;
Virtual machine

**Abstract** A cloud computing environment offers a simplified, centralized platform or resources for use when needed at a low cost. One of the key functionalities of this type of computing is to allocate the resources on an individual demand. However, with the expanding requirements of cloud user, the need of efficient resource allocation is also emerging. The main role of service provider is to effectively distribute and share the resources which otherwise would result into resource wastage. In addition to the user getting the appropriate service according to request, the cost of respective resource is also optimized. In order to surmount the mentioned shortcomings and perform optimized resource allocation, this research proposes a new Agent based Automated Service Composition (A2SC) algorithm comprising of request processing and automated service composition phases and is not only responsible for searching comprehensive services but also considers reducing the cost of virtual machines which are consumed by on-demand services only.


## 1. Introduction

Cloud computing is a business model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be elastically provisioned on demand through world wide web and released with minimal management effort or service provider interaction. From the vast number of resources available on the cloud, end user is required to pay only for services provided by a concerned service provider. There are a number of virtual machines (Ezugwu et al., 2013) present at cloud datacenter and each virtual machine handles one resource with many instances, respectively and since the resources are used at the request of end user, therefore the cost of usage increases automatically which becomes a major bottleneck in the deployment of too many virtual machines. With unlimited number of resources at cloud data center, allocating and discovering active and most suitable service resource is another major challenge. Besides various pros and cons associated with this

* Corresponding author. Tel.: +91 9896710680.
E-mail address: singh2208@gmail.com (A. Singh).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.jksuci.2015.09.001
1319-1578 

technology, enterprises are willing to execute their businesses by shifting either to public, private or hybrid cloud where public cloud offers the services and infrastructure off-site over the Internet, private cloud maintains the services and infrastructure on a private network and hybrid cloud includes a variety of public and private options with multiple providers adding to the cost of multiple security providers also. In contrast to public clouds which is known to be the most efficient in sharing resources, private clouds are more efficient in terms of security adding toward a high cost (Su et al., 2013) of maintaining the software and infrastructure. In order to manage the aforementioned cloud centers, there exist stringent requirements and the management of the same becomes complicated. Hence the need of a novel strategy for resource management is quite apparent. This work argues that mobile agents can potentially manage resource allocation, especially in distributed applications while considering request processing, automated service composition and cost optimization of virtual machines as primary factors. The current work proposes an autonomous service composition framework relaxing the management requirements to a large extent and hence overcoming the abovementioned cons. Next subsection provides a brief overview of networking architecture associated with typical data centers.

### *1.1. Data center networking architecture overview*

A cloud data center mainly comprises of servers, infrastructure, power draw or power supporting devices (UPS, generators etc.), cooling devices and networking components (Lenk et al., 2009). Each of these components has some cost involved with them. Cost involved with server, infrastructure (Marrone et al., 2013) and power supporting devices and cooling devices in not of direct concern for the user, however networking components are the backbone of accessing cloud services for the user. Thus, networking is of importance for end users and service providers. When user's task is computation intensive and requires resources from more than one server then the speed of computation may drop because of an increase in propagation delay between servers. Further, cost of communication and cost of service would increase with increase in distance between data centers. This is due to the fact that cost of WAN is significantly more than the cost of LANs. Thus the overall cost of providing service to the user will increase if the service involves use of physically distributed servers. Use of micro data centers has been suggested to improve efficiency of services to the users. Further agility (Greenberg et al., 2009) is the key to reduce cost of cloud services for users which is the ability to grow and shrink resources to meet demand of the user and to draw those resources from optimal locations.

The existing physical structure of data centers causes hindrance in optimal utilization of available resources. Fig. 1 given below illustrates physical architecture of a data center, it is taken from CISCO Systems (2004).

In a data center requests arriving from the Internet are at layer 3 and they are identified using an IP address (layer 3), they are routed through border and access routers to a layer 2 domain based on the destination virtual IP address (VIP). The VIP is configured onto the two load balancers connected to the top switches, and complex mechanisms are used to ensure that if one load balancer fails, the other picks up the traffic. For each VIP, the load balancers are configured with

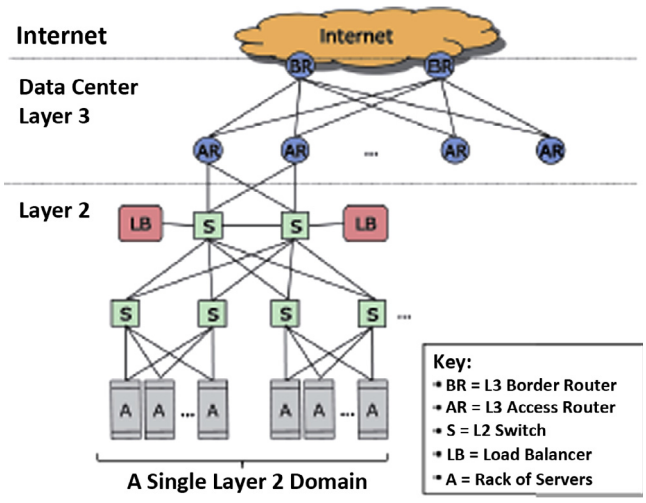


**Figure 1** Cloud data center networking architecture as suggested by CISCO (CISCO Systems, 2004).

a list of direct IP addresses (DIPs), which are the private and internal addresses of physical servers in the racks below the load balancers. This list of DIPs defines the pool of servers that can handle requests to that VIP, and the load balancer spreads requests across the DIPs in the pool.

For efficient service provisioning in clouds, user requests should be allocated to any server in one data center or other data centers, however present network architecture does not support agility to that extent and leads to issues such as fragmentation of resources and poor server to server connectivity. Researchers are making efforts to improve these networking hurdles (Greenberg et al., 2008) however these problems still prevail. This work aims to propose an intelligent service composition and provision mechanism so that prior to allocation of resources, optimal hardware resources may be allocated to user and some of networking problems may be avoided.

Rest of the paper is structured as follows. Section 2 discusses the related work in this field. Section 3 describes the proposed technique, algorithms and flow diagram based on it. Section 4 elaborates on the results and comparisons with the existing techniques. Finally conclusion is given in Section 5.

## 2. Related work

The section throws light on the work of some renowned researchers who had been pillars and founders of the current research work.

Research on resource management strategies in different fields (Chia-Ming et al., 2014) of distributed computing with different policies is not new. However in CC, dynamic resource provisioning (Quang-Hung et al., 2014) without delay or any compromise on delay is of utmost concern. Since, ubiquity and cost-effectiveness are two keywords describing CC, cost effectiveness centers on optimal resource allocation. Literature has been reviewed to explore existing strategies of resource allocation and scope of improvement. Buyya et al. (2002, 2003) presented resource allocation frameworks which could optimize the objective function for users and resource providers. Li et al. (2009) offered scheduling and optimization

techniques based on Service Level Agreement (SLA) ignoring the throughput and response time of data centers. Bennani and Menasce (2005) presented a predictive multi-class queuing network model for computing the mean response time but the model was not good enough to evaluate the cost in case server switches from one application to another. Singh et al. (2015) have presented an agent based load balancing mechanism. Arfeen and his coworkers (Arfeen et al., 2011) focused on network awareness and consistent optimization of resource allocation strategies and highlighted the research issues prevailing in this field. Zhang et al. (2010) emphasized that more efforts are required to make the existing performance models predictive and responsive. Zheng et al. (2009) proposed a binary integer programming method to solve independent optimization but linear problems only and is not suitable for dynamic and complex problems. Also, a few authors (Christodoulou et al., 2007; Doulamis et al., 2007) had proposed the game theoretic method to solve the optimization of resource allocation in network systems from the resource providers' perspectives. Ji et al. (2014) proposed a job scheduling algorithm based on greedy approach. Authors have implemented their algorithm in cloud environment and indicated success in reducing completion time of a task. Their implementation divides the tasks based on completion time and bandwidth requirements. However, in case resources are not found in a particular data center, this issue has not been paid attention. Hassan and Alamri (2014) proposed a resource allocation mechanism based on Nash Bargaining system for multimedia cloud computing, their scheme provides dynamic resource allocation with reduced cost. Authors have compared their algorithm with greedy approach of migration in case of overloaded *VM*s and indicated better results. However there is no bargaining for resource utilization. Marrone and Nardone (2015) proposed a model driven approach for resource allocation. Authors have deployed an automatic negotiation model using UML and Bayesian Network modeling approach. This works is completely based on negotiations. However, response time and cost optimization have been left unattended. Xiao et al. (2013) has introduced concept of skewness to measure unevenness in multi-dimensional resource utilization of a server. Different types of workloads can be combined to minimize skewness and improve overall server resource utilization. This mechanism provides overload avoidance while concerning green computing. Yee-Ming and Hsin-mie (2010) provided an allocation and pricing mechanism as a market-based model for allocating resources in a cloud computing environment. But this model is also not able to handle large scale problems adequately. Many more resource allocation mechanisms are available in Buyya et al. (2008), Jung and Sim (2011), Stoesser et al. (2007), Streitberger et al. (2007) which reflects that substantial efforts had already been put toward resource management in cloud computing but to the best of our knowledge none has proved to be suitable under all conditions. From the literature review it is clear the main purposes of scheduling algorithms are to minimize the resource starvation and to ensure the effective and fair resource scheduling (Singh and Malhotra, 2013). In fact, optimal resource allocation strategies have always been of utmost concern for researchers and hence the need to pay more attention on the resource scheduling policies is chirping in. Traditionally optimal resource allocation makes use of the Hungarian algorithm, which can work only on symmetric number of resources and requests. But, cost sharing model of CC deploys multi-tenancy, thus resource scheduling cannot be optimized using the Hungarian method always. This gave us motivation for the present work which focuses on an intelligent agent-based automated scheduling and service composition framework for cost optimization of resource provisioning in cloud computing. Next section elaborates the proposed framework.

## 3. The A2SC framework

With the aim to reduce the complexity, A2SC offers a twin layered architecture (see Fig. 2) where each layer can operate autonomously and cooperatively as well. Layer 1 named as Automated Request Processing Layer (ARPL) is responsible for initial request processing and locating the suitable service providers whereas layer 2 known as Automated Service Composition Layer (ASCL) mainly deals with service composition for which it might need to interact with layer 1. Each layer is equipped with a varying set of mobile agents and the functionality of all operational agents as delineated in Table 1 given below and levels i.e. ARPL and ASCL are discussed in the subsequent sections.

Fig. 3 given below provides interaction between various agents.

Upcoming subsections provide details of both layers of proposed mechanism.

### *3.1. Automated Request Processing Layer (ARPL)*

Primarily interface and assistant agents are the two active agents executing at this level and are shown in Fig. 4. The end user sends the resource request to data center having all potential and available resources. On every incoming request, *interface_agent* gets activated and immediately collects the relevant information required for processing a request further. If the request is found to be feasible requirement, it generates a request id ($\beta$) and then calls for an *assistant_agent* assigning the task of serving the request with respective id. *Assistant_agent* then explores the datacenter's *resource_repository* to find resources for request in hand. On finding suitable

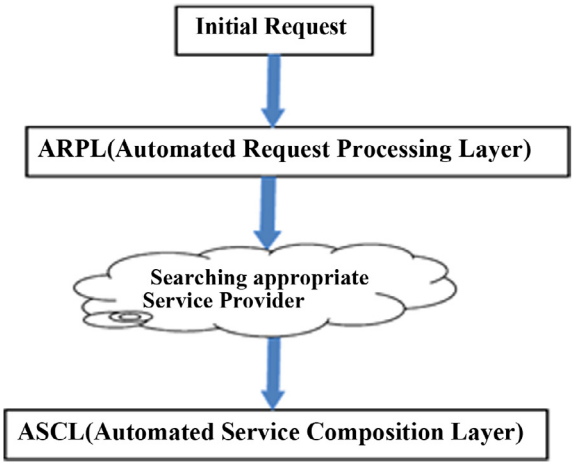


**Figure 2** Twin layered architecture of A2SC.

**Table 1** Functionality of operative agents.

| Sr. No. | Agent | Functionality |
|---|---|---|
| 1. | Interface agent | Interface Agent maintains the log of all requests received from users and further associates an Assistant Agent with every request. Also, it keeps the record of all virtual machines available for utilization along with the services specifications |
| 2. | Assistant agent | The respective assistant agent searches the resources from the available resource instances of current data center being maintained by Interface Agent. It also keeps the cost of available resources and contains an index of all unprocessed and processed requests along with request Ids |
| 3. | Broker agent | Broker Agent is a third party agent that acts as an intermediary between consumer and service provider in cloud computing. Its main role is to save the service searching time of consumer and provide the information of best vendors to the customers. Broker agent composes the contracts with providers on behalf of its customers |
| 4. | Directory agent | At the time of creation of data centers, enterprises are required to register all deployed agents with Directory Agent as this is the only agent that keeps record of all cloud agents including interface, assistant and broker agent and maintains their status such as available or allocated. It also maintains the record of its allocation delay time (ADT) where ADT is defined as time taken for allocation of services by the datacenter and also tracks the workload on respective data centers |
| 5. | Resource manager agent | Resource Manager Agent receives the user requests and corresponding resources recommended by Interface Agent and requests by user. It plays the final and important role in the twin layer architecture by allocating the suitable resources to every corresponding request |

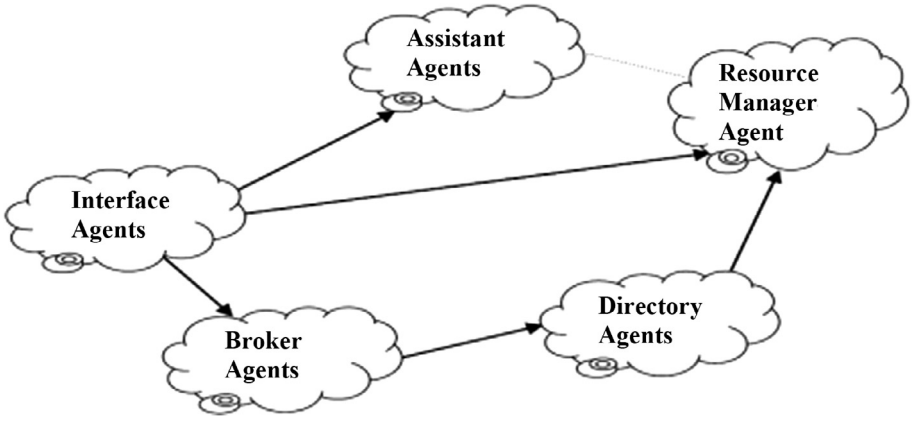


**Figure 3** Interaction between agents.

resources *assistant_agent sends* <req_id, list of resources> to *resource_manager_agent* where *resource_manager_agent* maintains the log of all available as well as allocated resources ($\alpha_i$).

Since the focus of this mechanism is to optimize cost while composing service for the user, cost of all resources allocated to the user should be less than or equal to the cost expected by the user in SLA. Eq. (1) given below represents user request as a group of instances of various resources as alloc_$\alpha_i I_i$ where alloc_$\alpha_i I_i = \{\alpha_1 I_2 + \alpha_2 I_3 + \cdots + \alpha_n I_2\}$. There must be some cost associated with this set represented as $\zeta_i(\text{alloc\_}\alpha_i I_i)$, this cost must be less than or equal to the cost specified by user in its request i.e. $\zeta_i(\beta_i)$. Thus at this stage every service provider must satisfy the following goal:

$$\alpha_i I_i \Longleftrightarrow \zeta_i(\alpha_i I_i) \leqslant \zeta_i(\beta_i) \quad \forall 0 < \beta_i < \alpha_i \text{ and } i = 1, 2, \ldots n \tag{1}$$

where $\alpha_i$ represents the total number of resources available and $\beta_i$ represents the number of requests by end user. $\alpha_i I_i$ is the resource instance matrix which represents the instances of every resource and $\zeta_i$ describes the cost of resource.

$$\text{Resource Instance Matrix} = \begin{matrix} \alpha_1 \\ \alpha_2 \\ - \\ \alpha_n \end{matrix} \begin{pmatrix} I_1 & I_2 & \cdots & I_n \\ I_1 & I_2 & \cdots & I_n \\ - & - & \cdots & - \\ I_1 & I_2 & \cdots & I_n \end{pmatrix}$$

$$\text{Cost of each resource} = \begin{matrix} \alpha_1 \\ \alpha_2 \\ - \\ \alpha_n \end{matrix} \begin{bmatrix} \zeta_1 \\ \zeta_2 \\ - \\ \zeta_n, \end{bmatrix}$$

If the condition depicted by Eq. (1) is satisfied, then *interface_agent* forwards all the terms and conditions such as cost, time etc. to the requester and waits for a response for a threshold time only. If the response is received within this threshold time, *interface_agent* processes the request accordingly; however, if user fails to respond, it is assumed that user is no more interested in the services of data center. Here, the requester may opt to bargain for new services or resources not listed in the request–responses and this calls for the execution of next level i.e. ASCL described in the upcoming section. Further, on

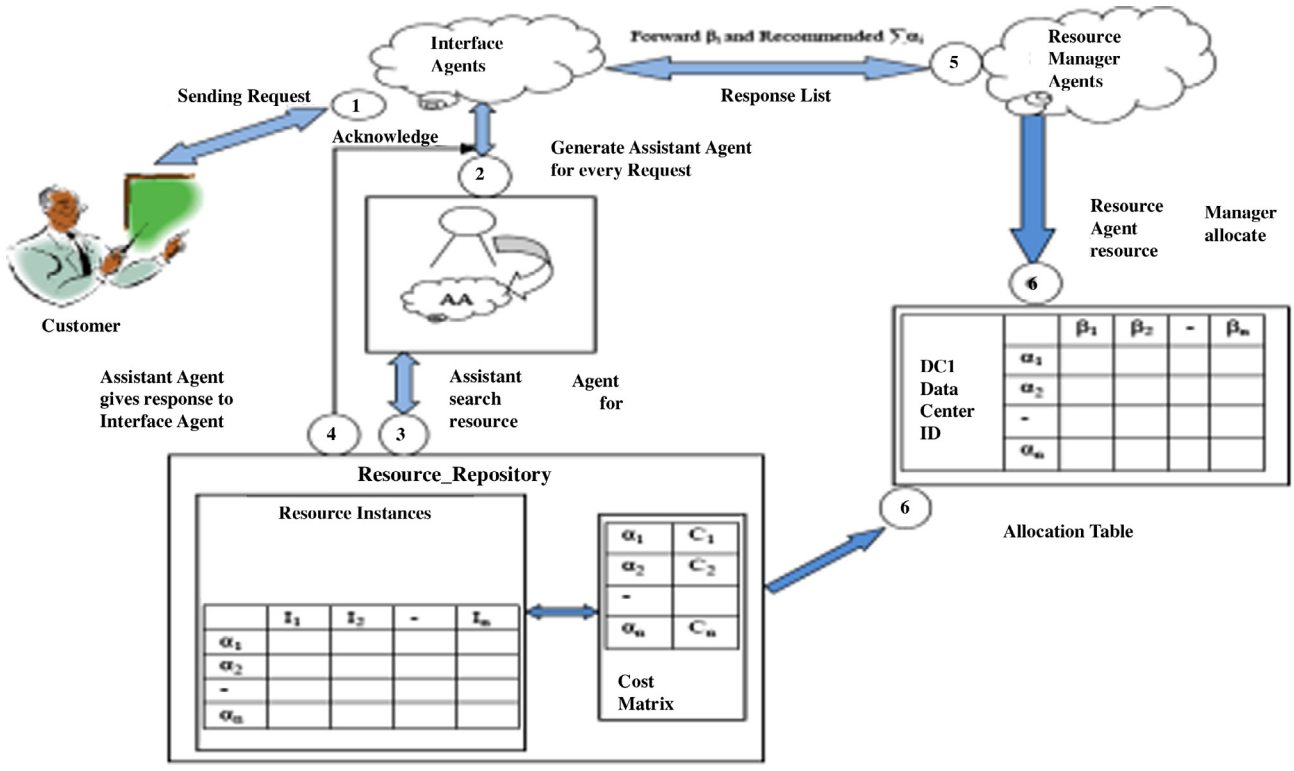


**Figure 4** Automated Request Processing Layer.

**Table 2** Description of different cases.

| Sr. No. | Cases | Description |
|---|---|---|
| 1 | $\sum \alpha_i = \beta_i$ | It means the desired resources are equal to the number of requests and are available in the data center |
| 2 | $\sum \alpha_i > \beta_i$ | It means the numbers of desired resources are less than available resources and there is resource availability in the data center |
| 3 | $\sum \alpha_i < \beta_i$ | It means the desired number of resources or resource instance are not available in the data center |

the basis of the values $\alpha_i$, $\beta_i$, *assistant_agent* determines if the current request can be processed. There are three possible cases as shown in Table 2 below:

All these cases deploy different course of action as discussed below:

Case I: $\sum \alpha_i = \beta_i$

Conventionally, the Hungarian algorithm[1] is exploited by data centers in the cases where request is equal to availability assuming that each person (here person implies resource) can do one job at a time though with varying degree of efficiency. But this significant characteristic i.e. one-to-one allocation becomes a major bottleneck in case of cloud computing. It fails to satisfy in many cases where same request can be satisfied by a lesser number of instances. The major contribution of this work is to propose a modified version of the Hungarian algorithm so that the same can be applied in an optimal way and hence satisfying the user with lesser instances rather than one-to-one allocation. The matrix should be based on instances of resource so that all instances may be properly utilized. If some of the instances have some space for more acquisition, then it would be acquired by some user demand. The below matrix represents the allocated instance of resources to its corresponding request. Let us suppose the defined matrix as follows:

Resource Instance Matrix

$$= \begin{array}{c} \\ \alpha_1 \\ \alpha_2 \\ - \\ \alpha_n \end{array} \begin{array}{c} \begin{array}{cccc} \beta_1 & \beta_2 & \cdots & \beta_n \end{array} \\ \begin{pmatrix} \{I_1\beta_1, I_2\beta_1\} & \{I_3\beta_2, I_2\beta_2\} & \cdots & \{I_n\beta_n\} \\ \{I_2\beta_1, I_2\beta_1\} & \{I_3\beta_1, I_4\beta_1\} & \cdots & \{I_5\beta_n\} \\ - & - & \cdots & - \\ \{I_1\beta_1, I_3\beta_1\} & \{I_2\beta_2, I_5\beta_2\} & \cdots & \{I_n\beta_n\} \end{pmatrix} \end{array}$$

Now *resource_manager_agent* has the objective (Wang et al., 2014) to minimize the cost with some constrains like minimum response time and optimal utilization of every instance. Eq. (2) given below indicates this objective function:

$$\text{Minimize } \zeta = \sum_{i=0}^{n} \alpha_i I_i \tag{2}$$

Subject to constraints

$$\beta = \sum_{i=0}^{n} I \quad \forall 0 < \beta_i < \alpha_i \tag{3}$$

i.e. requested resources should be less than available resources in the data center [refer Eq. (3)] and further the number of resources in data center should be more than 0 as shown in Eq. (4) below:

[1] http://en.wikipedia.org/wiki/Hungarian_algorithm.

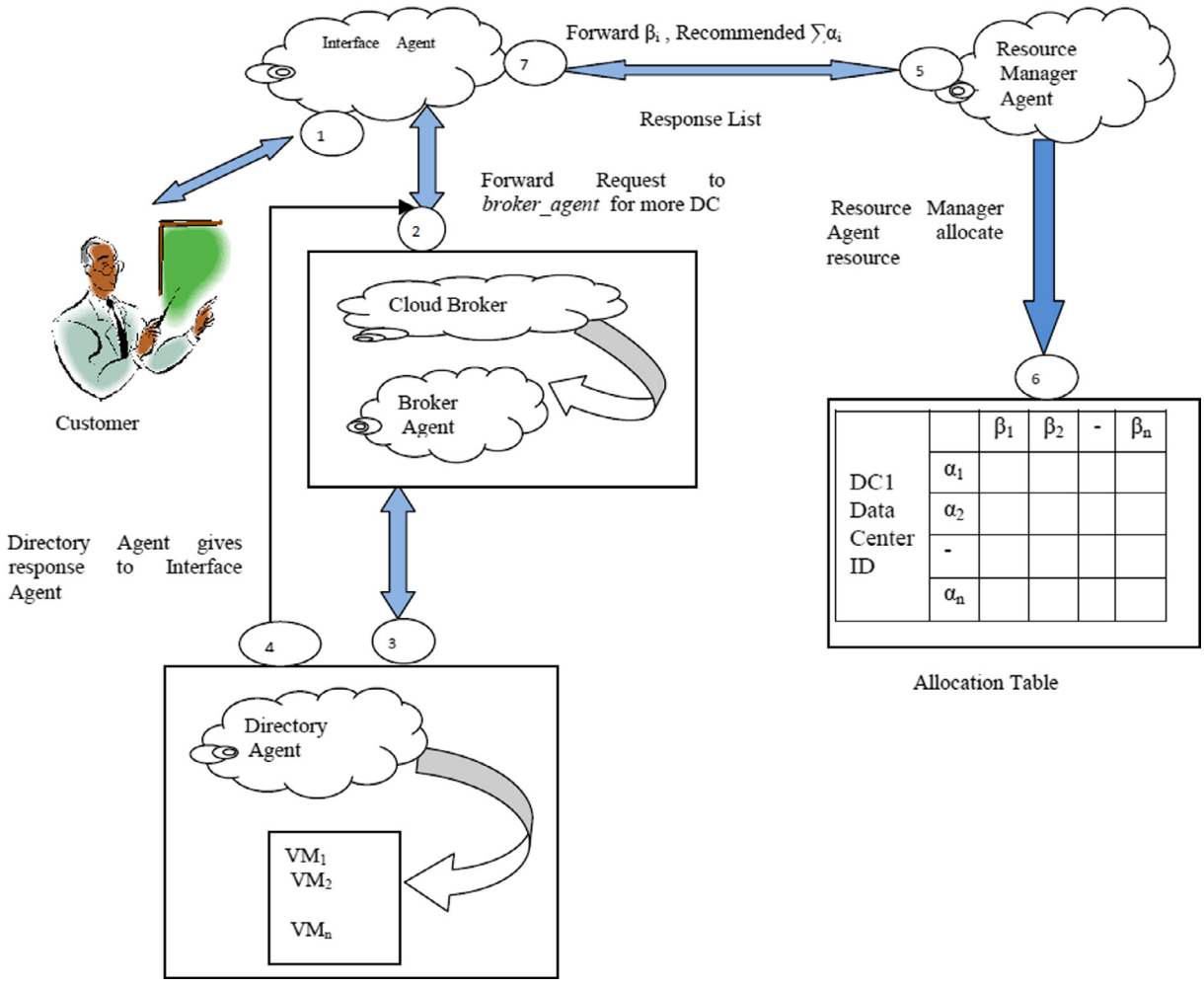


**Figure 5** Automated Service Composition Layer.

$$\alpha = \sum_{i=0}^{n} I \quad \forall 0 < \alpha_i \tag{4}$$

After applying this algorithm the *resource_manager_agent* will assign the resources to all requests and update the allocation table. Allocation table keeps the record of status of all resources. *Resource_manager_agent* sends information of allocated resources to the end user.

| Allocation table = | DC1 Data Center ID | | $\beta_1$ | $\beta_2$ | – | $\beta_n$ |
|---|---|---|---|---|---|---|
| | | $\alpha_1$ | | | | |
| | | $\alpha_2$ | | | | |
| | | – | | | | |
| | | $\alpha_n$ | | | | |

Case II: $\sum \alpha_i > \beta_i$

In case the numbers of requests are less than the number of resources, the resources will be assigned on FCFS basis satisfying the user demands.

Case III: $\sum \alpha_i < \beta_i$

In all other cases, requests will exceed the available resources. Now, the resource allocation can be done in two ways. On one hand the *interface_agent* may suggest the end users to modify their specifications based on BDI (belief–desire–intention) architecture.[2] The architecture is based on the set of beliefs i.e. the goals which a system can actually achieve, set of desires which are primarily the requests placed and intentions are the actions that are executed to achieve those beliefs and desires. Here, if the set of beliefs do not match with the set of desires, the intentions pertaining to the modification of desires are executed. The process is repeated till the goal is reached or refused by the client. On a similar note, *interface_agent* suggests the end user to modify/reduce the number of requested resources so that resources can be allocated as per the above two cases. Accordingly, the *interface_agent* would now arrange the resources at its own data center. In contrast to the above strategy, scalability (Singh and Malhotra, 2012) is one of the other options. It provides an option of moving from one cloud to another which is carried out with the help of *broker_ agent*. Scalability leads to the call for next layer i.e. Automated Service Composition Layer (ASCL).

[2] http://en.wikipedia.org/wiki/Belief%E2%80%93desire%E2%80%93intention_software_model.

### 3.2. Automated Service Composition Layer (ASCL)

When a request is received at this layer, *broker_agent* gets activated and is responsible for establishing a new contract between customer and service provider through service composition layer. It keeps the record of all *directory_agents* which further provides information about all data centers' record of virtual machines or resources available along with the cost of every service, response time and workload (Malhotra and Singh, 2015) associated with these resources at respective data centers. Workload of a data center ($WL_i$) is calculated in terms of virtual machines that are engaged and is given by the Eq. (5) below:

$$WL_i = \sum WL_i / VM \tag{5}$$

where $\sum WL_i$ is the total workload of data center and $VM$ is the number of virtual machine at that data center and it is directly proportional to response time. If $\sum WL_i < VM_i$ then the DC is underloaded and vice versa. In fact, *broker_agent* is required to search the suitable data center with minimum response time and cost of virtual machines. On completion of search process, a signal is communicated to *interface_agent* which is further transferred to end user. If the end-user agrees to all terms and conditions, the contract is established and a message *contract_established* is communicated to *resource_manager_agent* for the final assignment of resources. Fig. 5 elaborates the procedure of ASCL.

### 3.3. Algorithms

Algorithms of all five agents deployed in proposed mechanism are given in Figs. 6.1–6.5 below:

Interface_Agent (IA)

```
Interface_Agent( )
 {
 Input: Receive request from user;
 Output: Generate assistant_agent;
{
If(response== accepted)
Establish_contract;
Call RMA( ) ◄—— <βi, ∑αi>;
Call Broker_Agent( );
}
 }
```

**Figure 6.1** *Interface_Agent.*

Assistant_Agent (AA)

```
Assistant_Agent( )
 {
Input:user_request;
 Output: List<∑αi, ∑ζi,specifications> ;
 {
Search ∑αi in resource_repository;
 If (βi == satisfied)
IA( ) ◄— < βi, ∑αi, ∑ζi,specifications>;
else
IA( ) ◄— βi_not_satisfy;
}
 }
```

**Figure 6.2** *Assistant_Agent.*

Broker_Agent (BA)

```
Broker_Agent( )
 {
 Input: <βi>;
Output: IA( ) ◄—— <DC_Id,ζi(βi)>;
 {
 On receiving request βi from IA( );
 Send DA( ) ◄—— βi;
 Receive <DC_id, response_time,
VM_specification> ◄—— DA( );
IA( ) ◄—— send<DC_id, response_time,
VM_specification>;
Wait_for (acceptance(βi) from IA( );
 On receiving acceptance (βi) ◄—— IA( );
DC_id ◄—— establish_contract;
RMA( ) ◄—— send contract_establishes (βi);
}
 }
```

**Figure 6.3** Broker_Agent.

Directory_Agent (DA)

```
Directory_Agent( )
 {
Input:<request_id, specification> ◄—— BA( );
 Output: List of DC's;
 {
 Select candidate DC's;
 Calculate WLi = ∑WLi/VMi ;
 Calculate response_time α WLi ;
 Select DC with min(response_time);
BA( ) ◄—— <DC_id, ∑αi, ∑ζi, response_time> ;
}
 }
```

**Figure 6.4** Directory_Agent.

Resource_Manager_Agent (RMA)

```
Resource_Manager_Agent( )
 {
 Input: Receive List< forwardβi , recommended_∑αi>;
 Output: Resource_Allocation;
{
 On receiving < forwardβi , recommended_∑αi>;
Update_Allocation_table;
}
 }
```

**Figure 6.5** *Resource_Manager_Agent.*

### 3.4. Workflow

See Fig. 7.

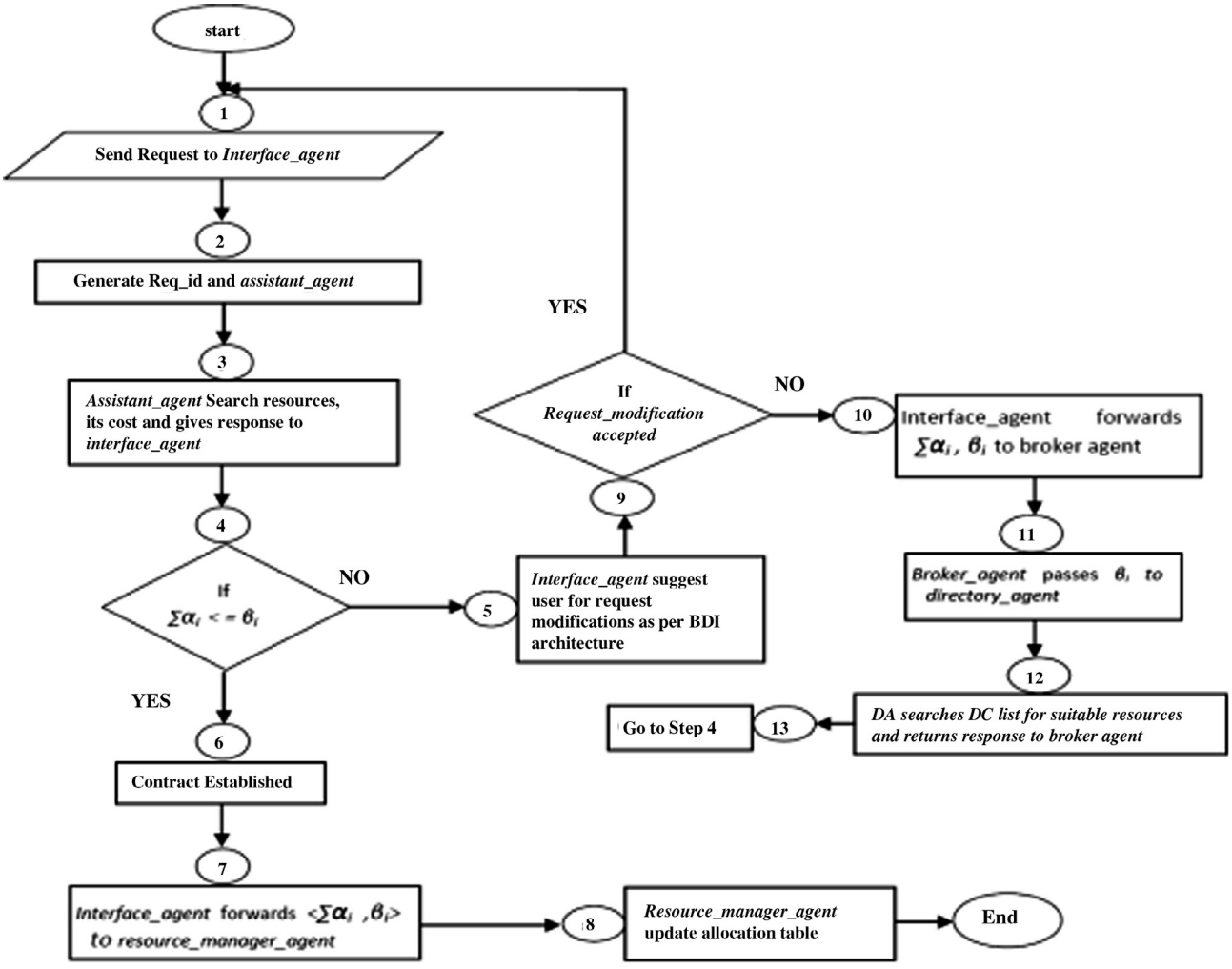


**Figure 7** Workflow of proposed A2SC framework.

**Table 3** Available resources and their respective elements.

| Data centers available | No. of available resource per data center | No. of instance per resource |
|---|---|---|
| Data Center – 1 | 4 {xp, ubuntu, win7, win8} | 4 |
| Data Center – 2 | 4 {xp, ubuntu, win7, win8} | 4 |
| Data Center – 3 | 4 {xp, ubuntu, win7, win8} | 4 |

Next section discusses the experimental outputs achieved.

## 4. Experimental results

This section presents the performance and result of proposed system in a small environment. The proposed mechanism being implemented is JAVA. Java is being chosen for this purpose because of its web based nature and support to agent technology. For the initial experiment we have considered three data centers having four resources namely XP, ubuntu, win7 and win8, respectively. We have assumed that there are four instances of each resource type. Table 3 given below provides the experimental setup.

Table 4 given below provides description of various resources in each data center. As discussed earlier how the assignment would be done, it is implemented in the code and afterward the corresponding response time taken for the resources allocation and also the cost for every instance assigned as per the user's request are calculated. This entire implementation has been done on a small scale which could be further enforced at a large scale using the appropriate technology and the required resources. We have taken different numbers of requests, for the same as well as different resources from that available in data centers and recorded the response time and cost incurred.

Table 5 shows the results after implementation.

**Table 4** Description of available resources.

| Resources | XP | Win7 | Ubuntu | Win8 |
|---|---|---|---|---|
| *1st Data Center* | | | | |
| Instances | Dotnet | Dotnet | Java | Office |
| | Office | Office | Office | Java |
| | Java | VBasic | Dotnet | VBasic |
| | VBasic | Java | VBasic | Dotnet |
| *2nd Data Center* | | | | |
| Instances | Dotnet | Dotnet | Dotnet | Dotnete |
| | Php | Php | Php | Php |
| | Java | Oracle | mysql | Java |
| | Oracle | Office | Vbasic | mysql |
| *3rd Data Center* | | | | |
| Instances | Oracle | Java | Php | Oracle |
| | Vbasic | Dotnet | Oracle | Php |
| | Office | Php | Java | Java |
| | mysql | Office | Dotnet | Office |

**Table 5** Cost and response time as per the user's requests.

| Various cases | No. of requests | Response time | Cost calculation |
|---|---|---|---|
| $\sum \alpha_i > \sum \beta_i$ | (i) Requests = 2 | (i) 5.953 s | (i) $9 |
| | (ii) Requests = 1 | (ii) 3.762 s | (ii) $4 |
| | (iii) Requests = 3 | (iii) 6.703 s | (iii) $13 |
| $\sum \alpha_i < \sum \beta_i$ | 5 | 10.391 s | $20 |
| Requests not found in same data center | 2 | 10.656 s | $10 |

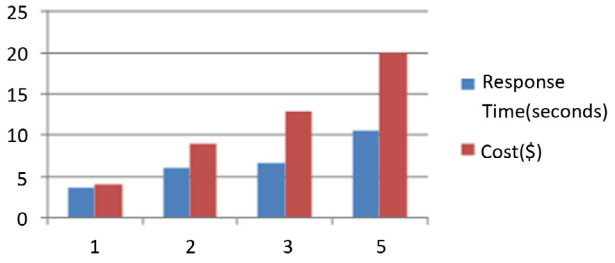


**Figure 8** Illustration of response time and cost estimation as per user's requests.

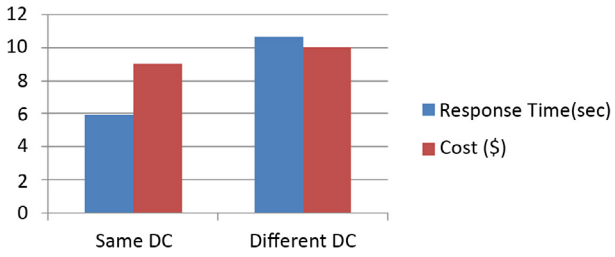


**Figure 9** Illustration of response time and cost estimation as per user's requests.

Fig. 8 illustrate the response time and cost of virtual machine as per the user request. As the number of requests would increase, the response time (in seconds) and total cost ($) also increased.

Fig. 9 indicates the difference between response time and cost, when the requests are allocated from same or different data centers. While repeating the experiment with varied number of resource requests it is achieved that when same instances are requested more, response time increases due to increase in total workload, however cost of resources does not increase, and even using scalability. Thus it is achieving the objective of cost optimization.

## 5. Conclusion

The work presented in this paper proposes a unique, intelligent and automated assignment strategy for assigning resources in cloud computing environment. In this mechanism various intelligent agents have been deployed to reduce system complexity by modularization. Broker agent facilitates search for optimal data center as per user requirements and service composition on user behalf till a contract is established between two parties or user enters a new service specification. Thus the proposed framework contributes toward eliminating user headache of finding optimal service provider in any situation and ensures efficient service allocation at the data centers. The proposed work also eliminates the limitation of the existing Hungarian algorithm and extends it by introducing two more cases and eliminating the one to one correspondence for resource allocation. The proposed strategy has also been implemented and results are found to be acceptable.

## Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.jksuci.2015.09.001.

## References

Arfeen, M.A., Pawlikowski, K., Willing, A.A., 2011. Framework for resource allocation strategies in cloud computing environment. In: Proceedings in 35th IEEE Conference on Computer Software and Application, pp. 261–266.

Bennani, M.N., Menasce, D.A., 2005. Resource allocation for autonomic data centers using analytic performance models. In: Proceedings of IEEE International Conference on Autonomic Computing, Seattle, pp. 192–208.

Buyya, R., Abramson, D., Giddy, J., Stockinger, H., 2002. Economic models for resource management and scheduling in grid computing. Concurr. Comput. Pract. Exp., 1507–1542 (Special Issue on Grid Computing Environments)

Buyya, R., Branson, K., Giddy, J., Abramson, D., 2003. The virtual laboratory: enabling molecular modeling for drug design on the world wide grid. Concurr. Comput. Pract. Exp. 15 (1), 1–25.

Buyya, R., Yeo, C.S., Venugopal, S., 2008. Market oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. In: Proceedings of 10th IEEE International Conference on High Performance Computing and Communications, pp. 234–242.

Wu, Chia-Ming, Chang, Ruay-Shiung, Chan, Hsin-Yu, 2014. Green energy-efficient scheduling algorithm using the DVFS technique for cloud datacenters. Future Gener. Comput. Syst. 37, 141–147.

Christodoulou, G., Koutsoupias, E., Vidali, A., 2007. A lower bound for scheduling mechanisms. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 1163–1169

CISCO Systems, 2004. Data center: Load balancing data center services.

Doulamis, N., Doulamis, A., Litke, A., Panagakis, A., Varvarigou, T., Varvarigos, E., 2007. Adjusted fair scheduling and non-linear workload prediction for QoS guarantees in grid computing. Comput. Commun. 30, 499–515.

Ezugwu, Absalom E., Buhari, S.M., Junaidu, S.B., 2013. Virtual machine allocation in cloud computing environment. Int. J. Cloud Appl. Comput. (IJCAC) 3 (2), 47–60.

Greenberg, A., Lahiri, P., Maltz, A.D., Patel, P., Sengupta, S., 2008. Towards a next generation data center architecture: scalability and commoditization. In: PRESTO'08, Seattle, Washington, USA.

Greenberg, A., Hamilton, J., Maltz, A.D., Patel, P., 2009. The cost of a cloud: research problems in data center networks. ACM SIGCOMM Comput. Commun. Rev. 39 (1), 68–73.

Hassan, Mohammad Mehedi, Alamri, Atif, 2014. Virtual machine resource allocation for multimedia cloud: a Nash bargaining approach. Procedia Comput. Sci. 34, 571–576.

Li, Ji, Feng, Longhua, Fang, Shenglong, 2014. An greedy based job scheduling algorithm in cloud computing. J. Software 9 (4), 921–925.

Jung, G., Sim, K.M., 2011. Agent based adaptive resource allocation on the cloud computing environment. In: Proceedings of IEEE Conference on Parallel Processing Workshop, pp. 345–351.

Lenk, A., Klems, M., Nimis, J., Tai, S., Sandholm, T., 2009. What's inside the cloud? An architectural map of the cloud landscape. In: Proceedings of the ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD '09), pp. 23–31.

Li, J., Chinneck, J., Wodside, M., Litoiu, M., 2009. Fast scalable optimization to configure service system having cost and quality of service constraints. In: Proceedings of 6th IEEE International Conference on Autonomic System, Barcelona, pp. 159–168.

Malhotra Manisha, Singh Aarti, 2015. Adaptive framework for load balancing to improve the performance of cloud environment. In: IEEE International Conference on Computational Intelligence and Communication Technology, pp. 224–228.

Marrone, Stefano, Nardone, Roberto, 2015. Automatic resource allocation for high availability cloud services. Procedia Comput. Sci., 980–987

Marrone, S., Nardone, R., Tedesco, A., D'Amore, P., Vittorini, V., Setola, R., et al, 2013. Vulnerability modeling and analysis for critical infrastructure protection applications. Int. J. Crit. Infrastruct. Prot. 6 (34), 217–227.

Quang-Hung, N., Thoai, N., Son, N., 2014. EPOBF: energy efficient allocation of virtual machines in high performance computing cloud. In: Transactions on Large-Scale Data and Knowledge-Centered Systems XVI; LNCS. pp. 71–86. ISBN: 978-3-662-45946-1; 2014.

Singh, A., Malhotra, M., 2012. Agent based framework for scalability in cloud computing. Int. J. Comp. Sci. Eng. Technol. (IJCSET) 3 (4), 41–45.

Singh, A., Malhotra, M., 2013. A comparative analysis of resource scheduling algorithms in cloud computing. Am. J. Comp. Sci. Eng. Surv. 1 (1), 1–19.

Singh, A., Juneja, D., Malhotra, M., 2015. Autonomous agent based load balancing algorithm in cloud computing. Procedia Comput. Sci. 45, 832–841, Published in International Conference on Advanced Computing Technologies and Applications, Science Direct.

Stoesser, J., Roessle, C., Neumann, D., 2007. Decentralized online resource allocation for dynamic web service applications. In: Proceedings of 4th International Conference of Enterprise Computing, E-Commerce, and E-Service, pp. 425–428.

Streitberger, W., Eymann, T., Veit, D., Catalano, M., Giulioni, G., Joita, L., Rana, O.F., 2007. Evaluation of economic resource allocation in application layer networks a metrics framework. In: Organisation: Service-, Prozess-, Market-Engineering, pp. 447–494.

Su, Sen, Li, Jian, Huang, Qingjia, Huang, Xiao, Shuang, Kai, Wang, Jie, 2013. Cost-efficient task scheduling for executing large programs in the cloud. Parallel Comput. 39, 177–188.

Wang, Xiao li, Wang, Yuping, Cui, Yue, 2014. A new multi-objective bi-level programming model for energy and locality aware multi job scheduling in cloud computing. Future Gener. Comput. Syst. 36, 91–101.

Xiao, Z., Song, W., Chen, Q., 2013. Dynamic resource allocation using virtual machines for cloud computing environment. IEEE Trans. Parallel Distrib. Syst. (TPDS) 24 (6), 1107–1117.

Yee-Ming, C., Hsin-mie, Y., 2010. Autonomous adaptive agents for market-based resource allocation of cloud computing. In: Proceedings of the 9th International Conference on Machine Learning and Cybernetics, Qingdao, vols. 11–14, pp. 2760–2764.

Zhang, Q., Cheng, L., Boutaba, R., 2010. Cloud computing: state-of-the-art and research challenges. J. Internet Serv. Appl., 7–18

Zheng, Y., Vasilakos, V.A., Wei, G., Xiaong, N., 2009. A Game-theoretic Method of Fair Resource Allocation for Cloud Computing Services. Business Media, LLc, pp. 252–269, Published in Springer Science.