HAL
open science

# A Self-adaptive Agent-based System forCloud Platforms

Merzoug Soltane, Yudith Cardinale, Rafael Angarita, Philippe Rose, Marta Rukoz, Makhlouf Derdour, Okba Kazar

**To cite this version:**

Merzoug Soltane, Yudith Cardinale, Rafael Angarita, Philippe Rose, Marta Rukoz, et al.. A Self-adaptive Agent-based System forCloud Platforms. to remove, Nov 2018, Paris, France. hal-01835777

**HAL Id: hal-01835777**

**https://hal.science/hal-01835777v1**

Submitted on 11 Jul 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# A Self-adaptive Agent-based System for Cloud Platforms

Merzoug Soltane
University of El-Oued LINFI
El-Oued, Algeria
merzoug-soltane@univ-eloued.dz

Yudith Cardinale
Dpto. de Computación y T.I
Universidad Simón Bolívar, Venezuela
ycardinale@usb.ve

Rafael Angarita
LISITE Laboratory, ISEP Paris
Paris, France
rafael.angarita@isep.fr

Philippe Rosse
UPPA, LIUPPA
Anglet, France
philippe.roose@univ-pau.fr

Marta Rukoz
Université Paris-Dauphine
CNRS, Paris, France
mrukoz@dauphine.fr

Derdour Makhlouf
Tebessa University
Tebessa, Algeria
m.derdour@yahoo.fr

Kazar Okba
Biskra University
Biskra, Algeria
kazarokba@gmail.com

***Abstract*—Cloud computing is a model for enabling on-demand network access to a shared pool of computing resources, that can be dynamically allocated and released with minimal effort. However, this task can be complex in highly dynamic environments with various resources to allocate for an increasing number of different users requirements. In this work, we propose a Cloud architecture based on a multi-agent system exhibiting a self-adaptive behavior to address the dynamic resource allocation. This self-adaptive system follows a *MAPE-K* approach to reason and act, according to QoS, Cloud service information, and propagated run-time information, to detect QoS degradation and make better resource allocation decisions. We validate our proposed Cloud architecture by simulation. Results show that it can properly allocate resources to reduce energy consumption, while satisfying the users demanded QoS.**

## I. Introduction

Cloud computing is the most flexible computing model to provide on-demand platforms for accessing custom resources and applications in the form of several types of services. Most common services in the Cloud are Platform as a Service (PaaS), Infrastructure as a Service (IaaS), Software as a Service (SaaS), Data as a Service (DaaS). The constantly increasing number of such services on the Cloud arises new challenges for ensuring their correct use, while guarantying self-adaptivity property, especially to manage user requirements (e.g., ensure QoS) [7] and optimize the use of resources (e.g., by minimizing energy consumption [16]). Taking into account these aspects, resource allocation becomes a complex task in highly dynamic environments, because of the increasing number of managed resources, as well as the increasing number of users with different requirements.

Existing resource allocation strategies lead to two unwanted situations: (i) the waste of resources when the demand is more than the capacity and demands cannot be attended, and (ii) the waste of resources when the demand is less than the capacity. In the first case, there are users not satisfied; in both cases, there are idle resources that are misused. A self-adaptive approach can direct the system to an ideal situation in which capacity is dynamically adapted to the demand of resources.

The main aim of this work is to address the resource allocation in Cloud platforms, towards a better utilization of resources and satisfaction of users requirements, by offering a self-adaptive approach. We propose a Cloud computing architecture that benefits from the combination of the Cloud and multi-agent technologies. The self-adaptive multi-agent system integrated into the proposed Cloud architecture is inspired on our previous work presented in [2], [3]. This self-adaptive system is a *MAPE-K-based (monitoring, analyzing, planning, executing, and knowledge)* approach [8] to reason and act, according to QoS (e.g., time, price), Cloud service information, and propagated run-time information, to detect QoS degradation, failures, or unavailability and make better resource allocation decisions for all users requests. The agents application knowledge for decision making comprises offline precomputed global and local information, user QoS preferences, and propagated actual resource state information. We validate our proposed multi-agent based Cloud architecture by prototyping and simulation. Results show that it can properly allocate resources to reduce energy consumption, while satisfying the QoS demanded by users.

## II. Related Work

Self-adaptive and autonomous management of computing resources have been topics of research interest and development for many years. In this section, we present recent literature concerning self-adaptive systems and agent-based approaches for cloud computing.

### A. Self-adaptive system in Cloud computing

In [10], a self-adaptive approach for Service-Level Agreement (SLA) based service virtualization in Cloud environments is proposed. The pro-

posed architecture allows the interoperability of service executions in heterogeneous, distributed, and virtualized environments. Inter Cloud is a self-management and SLA handling approach proposed in [4]. It is a Cloud federation oriented provisioning environment, that offers just in-time, opportunistic, and scalable application services. The idea of Inter Cloud is to envision utility-oriented federated IaaS systems that are able to predict application service behavior for intelligent down- and up-scaling infrastructures. Though it addresses self-management and SLA handling, the unified utilization of other services like PaaS and SaaS are not studied. These SLA-based works demonstrate that the combination of negotiation, brokering, and deployment using SLA-aware extensions and autonomic computing principles are suitable for achieving reliable and efficient service operation in distributed environments. However, its application in resource allocation has not been proved.

An approach for self-adaptive and self-configurable CPU resource provisioning for virtualized servers using Kalman filter[1] is presented in [9]. That work deals with the integration of a Kalman filter into feedback controllers to dynamically allocate CPU resources to virtual machines hosting server applications, creating a new resource management scheme. The novelty of this approach is the use of the Kalman filter to track the CPU usage and update the allocations accordingly. Another approach based on monitored data is described in [18], which consists on a self-adaptive Cloud monitoring approach with on-line anomaly detection. The main contribution of this work is the design of a self-adaptive monitoring framework, that can efficiently collect monitoring data from various distributed systems deployed in a Cloud computing environment. The authors focused on a correlation-based method to select key metrics representing other metrics. The problem with these works relies on the limitation of their applications. They are not generic and applicable for all kind of resource allocation.

Other works focus on QoS compliant [5], [15]. In [5], it is proposed a distributed self-adaptive architecture based on the Edge Computing concept with container-based technologies, such as Docker and Kubernetes, to ensure QoS for time-critical applications. For each container, features of resources required for the host can be allocated upon monitoring data and operational strategies defined by end-users, application developer, and administrator. In [15], it is proposed a QoS-aware Cloud Service Selection approach using service clustering and self-adaptation to efficiently support service selection in which runtime changes in the services QoS are taken into account to adapt to changes. As our work, these studies focus on QoS satisfaction; however, they do not consider local and remote information for more accurate decisions.

Based on the *MAPE* model, concepts of autonomic computing for adaptive management of Grid Computing are leveraged in [11]. However, this approach does not consider the deployment and virtualization mechanism in Cloud datacenters.

### B. Agent-based Clouds

Agents are networked software entities that can do specific tasks on behalf of a user and have a degree of intelligence that allows them to perform parts of their tasks autonomously and to interact with their environment in a successfully way. Agents are characterized by important features such as autonomy, sociality, rationality, responsiveness, proactiveness, and mobility [12]. Some studies have been proposed to design Cloud platforms based on multi-agent systems [1], [6], [14]. In [6], it is proposed the integration of agent-based system and Cloud computing for smart objects. This approach is suitable to effectively model Cooperative Smart Objects (CSO). In particular, a CSO is a smart object able to sense, store, and interpret information. In [14], it is presented a mechanism to provide dynamic load balancing for Clouds based on autonomous agents. The proposed mechanism is based on Ant mobile agents and whenever the load of a Virtual Machine (VM) reaches a threshold value, it initiates a search for a candidate VM from other datacenters, reducing in this way the allocation time. This work only considers the workload of VMs and does not consider the physical machines. In [1], a multi-agent system is proposed to manage the Cloud resources, while taking into account the customers QoS requirements. In this work a VM migration occurs when its hosting physical machine is facing an overloading or under loading problem. This approach is the most similar to ours, however, our approach is autonomic self-adaptive, allowing controlling the user requirements depending on the system and agents states.

To overcome the limitation of existing works in the field of self-adaptive model for Cloud computing, we propose a novel approach for the management of resources in Cloud platforms. Our approach is based on a multi-agent system implementing the *MAPE-K* model to enable self-adaptation. Particularly, we enhance the Cloud computing architecture with agent technologies to tackle the energy of consumption problem.

## III. Agents for the Cloud

One of the foundations of autonomic computing is the *MAPE-K* model for building autonomous self-adaptive agents [8]. *MAPE-K*-based agents can

[1]Kalman filtering is an algorithm that uses a series of measurements observed over time and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone.

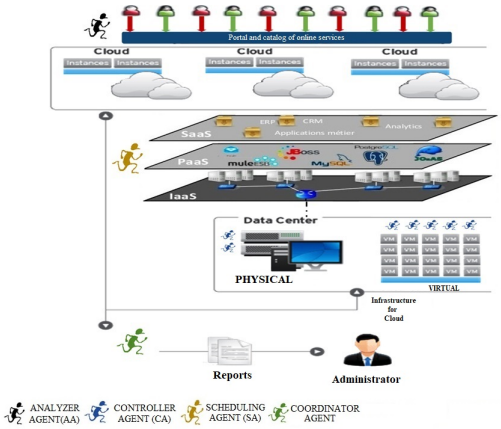


Figure 1. Agent-based Cloud Architecture.

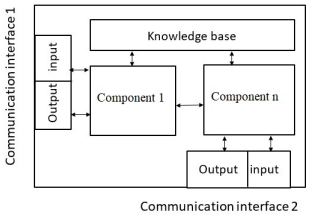


Figure 2. Agent Architecture Blueprint.

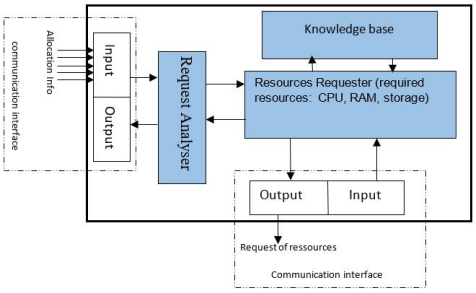


Figure 3. Analyzer Agent Architecture.

sense diverse properties of the environment and make *rational* decisions based on them. Such an agent can *monitor* changes on the environment where it exists, *analyze* the situation based on the sensed information, *plan* what it is going to do next based on this situation and its own capabilities and options, and *execute* the plan. All of these steps are supported by the *knowledge* the agent has about the environment and itself. This model is usually implemented as a loop to permanently sense the environment and adapt to changes.

Figure 1 shows our proposed general multi-agent Cloud Architecture, based on three layers: user interface, Cloud instances, and datacenter infrastructure. The user layer provides an interface to access the Cloud services and through which users specify their resource allocation needs. Analyzer Agents identify the resources and services demanded by users and build specific queries. Cloud instances layer represents an inter-layer between upper layer (i.e., users) and lower layer (i.e., datacenters). Scheduler Agents work at this layer. Datacenter layer provides resources as a service. It consists of a physical layer (e.g., servers, host, physical manage) and a virtual layer (i.e., instances of VMs). There exists a Controller Agent per each physical machine in datacenters. Additionally, the Coordinator Agent supervises the whole process. Figure 2 shows our proposed agent architecture blueprint. Internal agent components rely on a modular architecture, favoring the understanding and reuse of the main components of each agent. Each component focuses on a specific function required for accomplishing the purpose of the agent. Each agent can communicate with other agents through its communication interfaces. In this section we detail the analyzer agent, the scheduling agent, and the controller agent. The coordinator agent supervises the whole process.

### A. The Analyzer Agent

The analyzer agent is the first agent launched by the system. We illustrate its detailed architecture in Figure 3. Its goal is to study a user requirement (done by the Request Analyzer Component) and to build a request specifying the resources needed to satisfy it (through the Resources Requester component). This agent relies on a knowledge base containing individual knowledge and analytical knowledge, as follows:

**- Individual knowledge:** it reflects the agent self-knowledge, including the following:

- *name*: Agent analyzer (AA);
- *address*: the location where it is deployed; e.g., web Cloud service interface;
- *individual objectives*: its objective is analyze the resources requested by a user;
- *state*: it can have the following execution states: receiving user request, sending resource allocation request, and getting resource allocation response to/from the Scheduling Agent.

**- Analytical knowledge:** it concerns rules stored in its knowledge base dictating its behavior and actions.

Its detailed behavior is illustrated in the algorithm presented in Alg. 1. It receives the user requirement, which is analyzed by the Request Analyser component (lines 2 to 4 in Alg. 1), by identifying the required resources (line 5 in Alg. 1). This information is received by the Resources Requester module (line 9 in Alg. 1), which in turn build a request with the required resources to send it to the Scheduling Agent (lines 10 and 11 in Alg. 1).

### B. The Scheduling Agent

The Scheduling Agent is the central agent in the system. Its goal is to allocate resources needed by users. We illustrate its architecture in Figure 4. The Resource Allocator module receives requests from the Analyzer Agent and makes allocation decisions respecting the requested resources and QoS, and

```
Input  : (i), service description; (ii), SAL; (iii), id. user
Output : (i), needed resources
switch communication interface do
    case input interface do
        if service description then
            analyzerProcess(service description);
            return resources = { CPU, RAM, disk,... };
        end
    end
    case output interface do
        if analyzerProcess returns resources then
            request = resources + id.user;
            send request to the scheduling agent;
        end
    end
end
```

**Algorithm 1:** Analyzer Agent (AA)

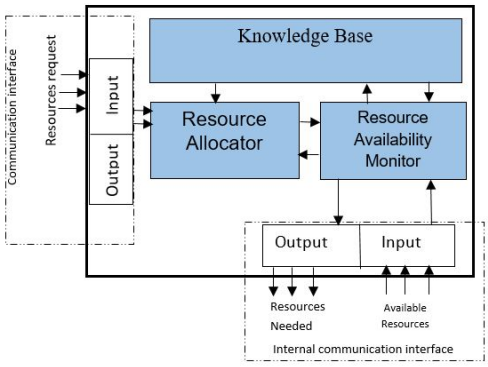


Figure 4. Scheduling Agent Architecture.

according the available resources. The Resource Availability Monitor is in charge of updating the resources state.

The knowledge base of the scheduling agent is composed by two parts:

**- Individual knowledge:** reflects the agent self-knowledge, including:

- *name*: Agent Scheduling (SA);
- *address*: the location where it is deployed; e.g., the resource management system in the data center;
- *individual objectives*: its objective is the resource allocation;
- *state*: this agent can have the following execution states: receiving request from the Analyzer Agent, sending a request of available resources, and waiting for a response to that request.

**- Allocation knowledge:**

- *Available resources*: the available resources on all physical machines;
- *Self-adaptive options*: for example, retry, resource replacement or reallocation.
- *Inference engine and rules*: the inference engine applies rules to the knowledge base to deduce new information. Rules can specify the actions to take given certain conditions.

Its detailed behavior is illustrated in Figure 5 and in the Alg. 2. If it receives a resource request, it checks the availability of those resources and

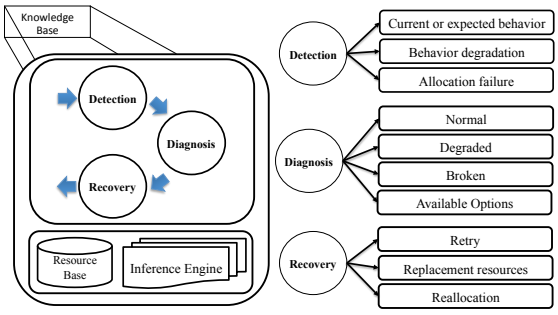


Figure 5. Scheduling Agent Behavior (General Description).

sends an allocation request to the Controller Agent (lines 3 to 5 in Alg. 2). When it receives from the Controller Agent the availability of resources, it updates the information in its knowledge base and does the reserves the resources (allocation process) for the corresponding query (lines 8 to 16 in Alg. 2). For the allocation process, the Scheduling Agent verifies the system state and the SLA (see Alg. 3). If the system state is $normal$ and the $SLA = acceptable$, it allocates the resources (lines 1 to 7 in Alg. 3); if the system state is $degraded$, but the $SLA = acceptable$, it allocates the resources (lines 8 to 12 in Alg. 3); otherwise it applies its rules and acts according the available options ( lines 13 to 17 Alg. 3).

```
Input  : (i), resources request; (ii), available resources
Output : (i), allocated resources
switch communication interface do
    case input interface do
        if resources request then
            check available resources;
            send request to the Controller Agent;
        end
    end
    case input interface do
        if analyzerProcess returns resources then
            TAB.resources = [null];
            while receiving response from the controller agent
              do
                TAB.resources = [CA1.resources,
                  CA2.resources,...];
                return TAB.resources;
                store resources' data in the knowledge base;
                //allocation process
                if available resources ≡ needed resources
                  then
                    Alg. 3
                end
            end
        end
    end
end
```

**Algorithm 2:** Scheduling Agent (SA)

### C. The Controller Agent

The goal of the controller agent is to manage datacenter resources, by tracking their status; through the Resource Status Updater component. Figure 6 illustrates its detailed architecture. The knowledge base of this agent contains the following information:

**- Individual knowledge:** reflects the agent self-knowledge, including:

- *name*: Controller Agent (CA);

```
1  switch system state do
2      case normal do
3          if SLA ≡ acceptable then
4              allocate resources;
5              return resources;
6          end
7      end
8      case degraded do
9          if SLA ≡ acceptable then
10             allocate resources;
11             return resources;
12         else
13             available options = (retry or replacement or
                 reallocation);
14             go to line 7;
15         end
16     end
17 end
```

**Algorithm 3:** Scheduling Agent (SA) state verification

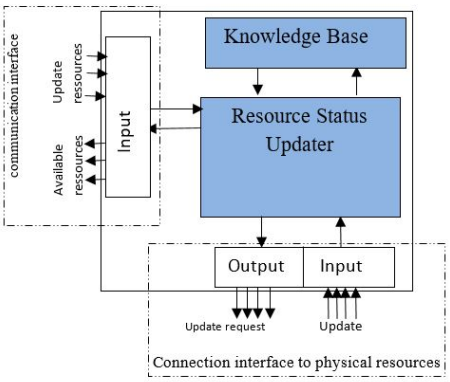


Figure 6. Controller Agent Architecture.

- *address*: the location where it is deployed; e.g., physical servers in the data center;
- *individual objectives*: the objective of this agent is to track resource status;
- *state*: this agent can have the following execution states: receiving the request from the Scheduling Agent and sending the available resources to Scheduling Agent.

**- Controller knowledge:** it refers to the information this agent has about the status of the different resources in the datacenter.

Finally, its detailed behavior is illustrated in the algorithm presented in Alg. 4. It permanently monitors the resources' states and updates this information in its knowledge base (lines 7 to 15 in Alg. 4) . When a request from the Scheduling Agent arrives, it responds with the resources' statuses (line 18 in Alg. 4).

### D. Resource allocation and inter-agent Interaction

The interaction among agents is illustrated in Figure 7. The interaction process starts when the Analyzer Agent receives a user request, which is analyzed, interpreted, sent it to the Scheduling Agent. The Scheduling Agent verifies the system state (`state == normal`) and sends a request to the Controller Agent asking for information about the free resources. As a reminder, the Controller Agent tracks the status of resources in the datacenter. When the Scheduling Agent gets the response from the Controller Agent, it verifies if the proposed QoS and available resources match the user requirements before making a final decision about resource allocation. Finally, the Scheduling Agent sends a detailed report about the process and decisions made to the Coordinator Agent that supervises the whole process. The possible agent states are *normal*, *degraded*, and *broken*. An agent is in the *normal* state if the execution goes as expected concerning QoS allocation and system failures. An agent is in the *degraded* state if there is a degraded QoS allocation or failures in the system, but the QoS allocation still falls within the allowed range. Finally, an agent is in the *broken* state if the QoS allocation is not acceptable or there are irreparable failures in the system.

```
   Input  : (i), Scheduling Agent's request
   Output : (i), available resources response
1  switch communication interface do
2      case input interface do
3          if Scheduling Agent's request then
4              update resources;
5          end
6      end
7      case physical output interface do
8          if connection.resources then
9              resources.stat = getNewStat(resources.stat);
10             return resources.stat;
11         end
12     end
13     case physical input interface do
14         get resources.stat;
15         store resources.stat's data in the knowledge base;
16     end
17     case output interface do
18         send resources.stat to Scheduling Agent;
19     end
20 end
```

**Algorithm 4:** Controller Agent (CA)

## IV. Illustrative Example

To illustrate our approach, we propose a scenario of resource allocation, shown in Figure 8. The steps of the allocation process are the following:

1) In the first step, a user requests the desired services and resources through a web interface, by expressing the services/resources needed and the expected QoS (e.g., desired resource capacity, acceptable range of capacity, acceptable time and price). The Analyzer Agent gets this request and studies the existing services/resources to extract the requirements concerning the needed resources and the expected QoS. Some examples of resources are CPU, RAM, and disk space. At the end of this step, the Analyzer Agent builds a query, called $Q$, specifying the needed resources and expected QoS, and forwards it to the Scheduling Agent.
2) In the second step, the Scheduling Agent extracts the resource requirements forwarded by the Analyzer Agent. Its goal is to prepare

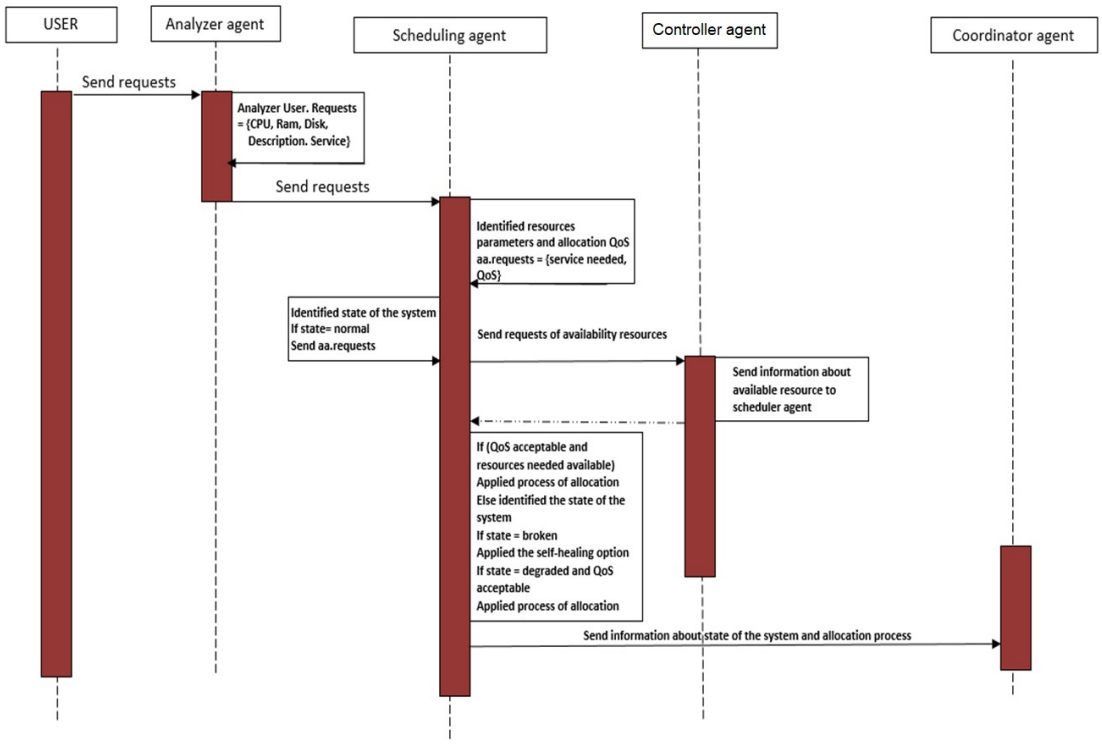


Figure 7. Inter-agent Interaction.

the necessary resources to satisfy the request $Q$, by building a Virtual Machine, called it $VM_Q$.

The request $Q$ forwarded by the Analyzer Agent is composed of three elements, as follows $Q = (R, C, QoS)$, where:

$R = (r_1, r_2, r_3, ..., r_n)$ represents resources and services needed by the user;

$C = (c_1, c_2, c_3, ..., c_n)$ is the capacity corresponding to each resource; and

$QoS = (qos_1, qos_2, qos_3, ..., qos_n)$ represents the QoS for each resource/service.

We represent the combination of resources and their corresponding requested capacities and QoS to build the suitable $VM_R$ as:

$$VM_R = \begin{pmatrix} r_1 & c_1 & qos_1 \\ r_2 & c_2 & qos_2 \\ \vdots & \vdots & \\ r_n & c_n & qos_n \end{pmatrix}$$

3) After the identification of requested resources, the Scheduling Agent sends a request to the Controller Agent to demand the corresponding free resources.
4) The Controller Agent receives the request from the Scheduling Agent and responds with the current state of the physical machines. There exists one Controller Agent, $CA_i$, for each physical machine, $PM_i$, in the Cloud. $CA_i$ manages the resources and information of its corresponding $PM_i$, including:
   - The set of virtual machines that run in $PM_i$, denoted as $V_i = (VM_1, VM_2, ..., VM_m)$;
   - The set of capacities corresponding to each virtual machine, denoted as $C_i = (c.VM_1, c.VM_2, ..., c.VM_m)$;
   - The set of available resources on in $PM_i$, denoted as $RA_i = (ra_1, ra_2, ..., ra_p)$;
   - The set of capacities corresponding to each available resource, denotes as $CA_i = (ca_1, ca_2, ..., ca_p)$.

   Then, the Controller Agent sends this information to the Scheduling Agent. We model the response of a Controller Agent $CA_i$ as two matrices:

$$T.PM_i =< \begin{pmatrix} VM_1 & c.VM_1 \\ VM_2 & c.VM_2 \\ \vdots & \vdots \\ VM_n & c.VM_n \end{pmatrix}, \begin{pmatrix} ra_1 & ca_1 \\ ra_2 & ca_2 \\ \vdots & \vdots \\ ra_p & ca_p \end{pmatrix} >$$

5) In the next step, the Scheduling Agent stores the information received from the Controller Agent in its knowledge base; then, it builds a global matrix of available resources at time $t$, denoted as $A(t)$:

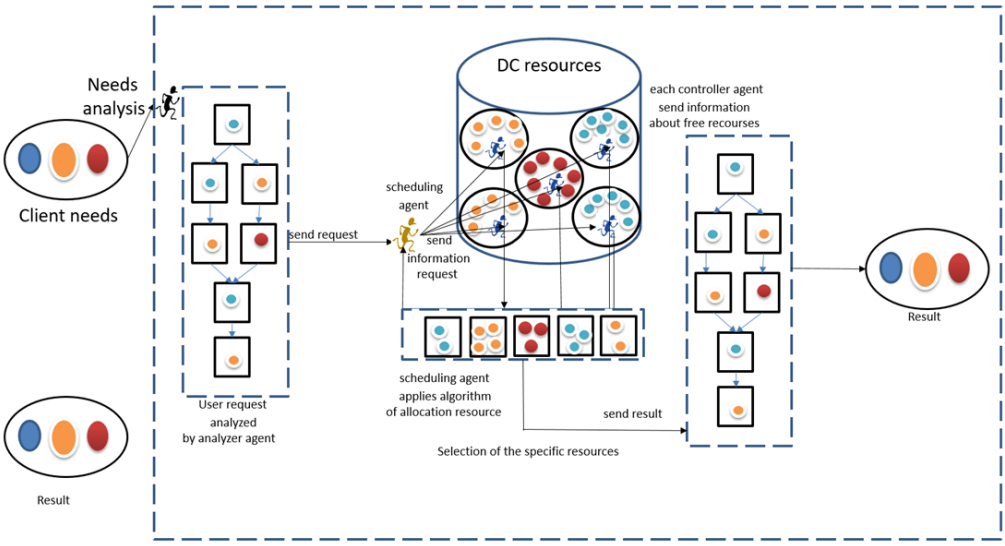


Figure 8. Allocation Scenario.

$$A(t) = \begin{pmatrix} PM_1 & RA_1 & CA_1 \\ PM_2 & RA_2 & CA_2 \\ \vdots & & \\ PM_n & RA_n & CA_n \end{pmatrix}$$

where $PM_i$ are physical machines and $RA_i$ and $CA_i$ are the set of available resources with their corresponding capacities.

6) Finally, after building the global resource matrix, the Scheduling Agent applies its self-adaptive allocation algorithm to build from $A(t)$ a $VM$ that satisfies $VM_Q$.

## V. Experiments

We measured the performance of our multi-agent system in terms of energy consumption, by simulation. To do so, we used OMNeT [17] and ICan Cloud [13] to simulate its functionalities. OMNeT is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. ICan Cloud is a simulation platform aimed to model and simulate Cloud computing systems. This simulation framework has been developed on the top of OMNeT++ and INET frameworks. The multi-agents system and the scheduling algorithm have been developed in C++, by using sockets to simulate their communication.

We simulated a scenario for allocating resources as mentioned in Section IV. Subsequently, we implemented a Cloud computing service provider to offers Cloud services to users. The data center implemented for this Cloud runs five heterogeneous servers. We randomly generated queries ($Q = (R, C, QoS)$) to submit to the Cloud in randomly time intervals (in seconds) and measured the aggregated energy consumption of the five servers.

Figure 9 shows an extract of the simulation running. Note that queries are submitted at different time intervals: the first one at time 0, the second one at 500ms, the third one at 5sec, and so on until the end of the simulation at 102min. All queries were satisfied in terms of needed resources and required QoS. Figures 10 and 11 show the energy consumption rates in Server 1 and Server 2, respectively. When we apply our self-adaptive strategy for resource allocation, we observe a remarkable change in the level of energy consumption, showing the efficiency of the scheduling algorithm. For Server 1 (Figure 10), the decrease of energy consumption was from 2500 megawatts to 480 megawatts, while for Server 2 (Figure 11), it was from 3500 megawatts to 600 megawatts. These decreases of energy consumption happen at times ranging from 0 minutes to 102 minutes.

These results put in evidence the performance of our algorithm of QoS allocation, confirming the choice of using a multi-agent system implementing a self-adaptive mechanism for the smart management of resources in Cloud data centers.

## VI. Conclusions

In this work, we tackled the problem of resource allocation in Cloud platforms. To that end, we have proposed a multi-agent architecture based on the *MAPE-K* model. For our future work, we plan to present a formal model of our approach defining the query for resource allocation, user preferences, agent rules, and state transitions. We also plan to describe in detail the scheduling algorithm and present additional experiments to evaluate other performance aspects, besides energy consumption.

## References

[1] Mahmoud Al-Ayyoub et al. Multi-agent based dynamic resource provisioning and monitoring for cloud computing

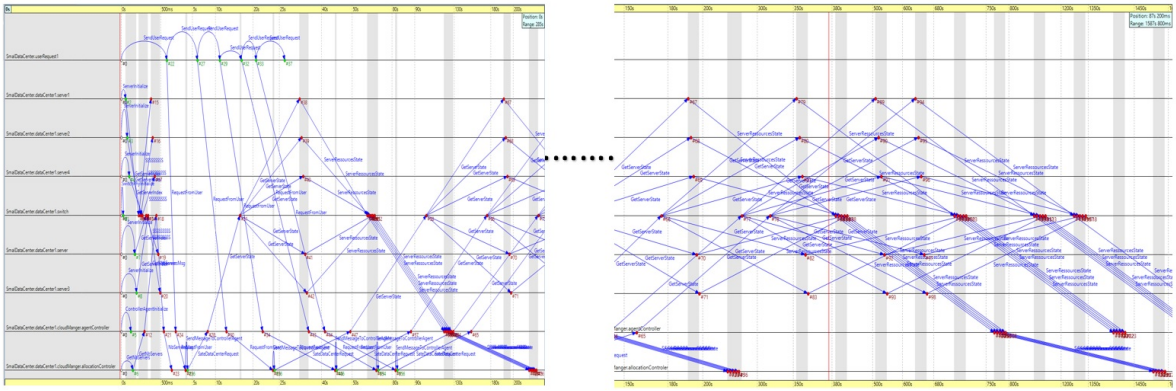

Figure 9. Extract of the Simulation Output.

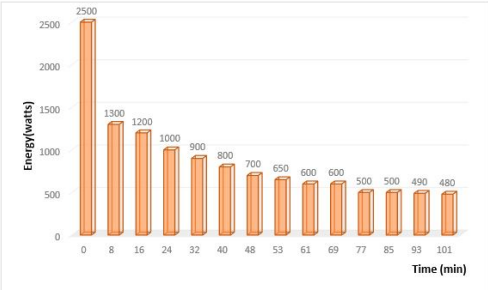


Figure 10. Energy consumption *Server 1*.

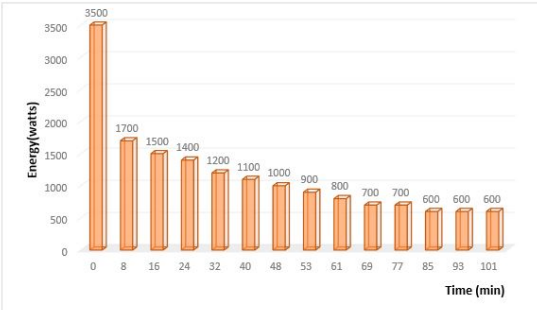


Figure 11. Energy consumption *Server 2*.


systems infrastructure. *Cluster Computing*, 18(2):919–932, 2015.

[2] Rafael Angarita, Marta Rukoz, and Yudith Cardinale. Modeling dynamic recovery strategy for composite web services execution. *World Wide Web*, 19(1):89–109, Jan 2016.

[3] Rafael Angarita, Marta Rukoz, Maude Manouvrier, and Yudith Cardinale. A knowledge-based approach for self-healing service-oriented applications. In *Proc. of International Conference on Management of Digital EcoSystems (MEDES)*, pages 1–8, 2016.

[4] Rajkumar Buyya et al. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In *Proc. of Algorithms and Architectures for Parallel Processing*, pages 13–31, 2010.

[5] Kieran Evans et al. Dynamically reconfigurable workflows for time-critical applications. In *Proc. of Workshop on Workflows in Support of Large-Scale Science*, pages 7:1–7:10, 2015.

[6] G. Fortino et al. Integration of agent-based and cloud computing for the smart objects-oriented iot. In *Proc. of Int. Conf. on Computer Supported Cooperative Work in Design (CSCWD)*, pages 493–498, 2014.

[7] J. Octavio Gutierrez-Garcia and Kwang Mong Sim. Agent-based cloud service composition. *Applied Intelligence*, 38(3):436–464, 2013.

[8] Didac Gil De La Iglesia and Danny Weyns. Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Trans. Auton. Adapt. Syst.*, 10(3):15:1–15:31, September 2015.

[9] Evangelia Kalyvianaki et al. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using kalman filters. In *Proc. of the Int. Conf. on Autonomic Computing*, pages 117–126, 2009.

[10] A. Kertesz, G. Kecskemeti, and I. Brandic. An interoperable and self-adaptive approach for sla-based service virtualization in heterogeneous cloud environments. *Future Gener. Comput. Syst.*, 32:54–68, March 2014.

[11] K. Lee et al. Adaptive workflow processing and execution in pegasus. In *Proc. of Internat. Conf. on Grid and Pervasive Computing - Workshops*, pages 99–106, 2008.

[12] M., J. A. Rodriguez-Aguilar, and J. Cerquides. A survey on sensor networks from a multiagent perspective. *The Computer Journal*, 54(3):455–470, March 2011.

[13] Alberto Núñez et al. icancloud: A flexible and scalable cloud infrastructure simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

[14] Aarti Singh et al. Autonomous agent based load balancing algorithm in cloud computing. *Procedia Computer Science*, 45:832–841, 2015.

[15] Giandomenico Spezzano. Using service clustering and self-adaptive mopso-cd for qos-aware cloud service selection. *Procedia Computer Science*, 83:512 – 519, 2016.

[16] Shekhar Srikantaiah et al. Energy aware consolidation for cloud computing. In *Proc. of Conference on Power-aware Computing and Systems*, volume 10, pages 1–5, 2008.

[17] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proc. of Int. Conf. on Simulation tools and techniques for communications, networks and systems & workshops*, page 60, 2008.

[18] Tao Wang et al. Self-adaptive cloud monitoring with online anomaly detection. *Future Gener. Comput. Syst.*, 80(C):89–101, March 2018.