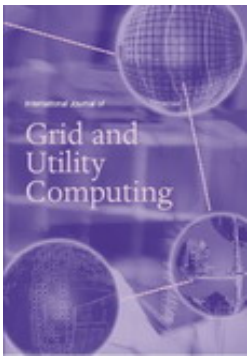


International Journal of Grid and Utility Computing

ISSN online: 1741-8488 - ISSN print: 1741-847X
https://www.inderscience.com/ijguc

An intelligent mechanism for requirements change management of cloud computing requests

Nassima Bouchareb

DOI: 10.1504/IJGUC.2024.10068152

Article History:

Received:	17 February 2023
Last revised:	22 September 2023
Accepted:	27 December 2023
Published online:	12 January 2025

An intelligent mechanism for requirements change management of cloud computing requests

Nassima Bouchareb

LIRE Laboratory,
Department of Software Technologies and Information Systems,
Faculty of New Technologies of Information and Communication,
University of Abdelhamid Mehri,
Constantine, Algeria
Email: nassima.bouchareb@univ-constantine2.dz

Abstract: Cloud environments are typically highly dynamic, making it challenging to identify, clarify and manage their requirements, especially when services and requirements change unpredictably. In this work, we introduce an agent-based mechanism designed to manage requirements changes, with a specific focus on requests for Virtual Machines (VM) allocation/reallocation in a Cloud Computing system. We begin by explaining the principle of the virtual machine allocation mechanism and the roles of the various agents involved. Next, we present the steps of the requirements change management process. Additionally, we delve into the various cases of requirements changes caused, whether initiated by the Cloud provider or customers, including adding a request, cancelling a request and modifying a request. Finally, we provide a case study and simulation results demonstrating that our proposed mechanism increases the number of accepted requests, enhances tasks processing and minimise energy consumption, ultimately boosting the profit of Cloud providers.

Keywords: cloud computing; virtual machine allocation/reallocation; requirements engineering; multi-agent systems; migration; energy efficiency.

Reference to this paper should be made as follows: Bouchareb, N. (2025) 'An intelligent mechanism for requirements change management of cloud computing requests', *Int. J. Grid and Utility Computing*, Vol. 16, No. 1, pp.15–28.

Biographical notes: Nassima Bouchareb is a Dr in the Department of Software Technologies and Information Systems at the University of Abdelhamid Mehri, Constantine 2, Algeria. She received her HDr/habilitation degree and PhD degree, in Computer Science from the University Constantine 2 – Abdelhamid Mehri, Algeria, in 2021 and 2015, respectively. She has benefited from two scholarships to France, in License at the University of Via Domitia, Perpignan, 2009 and Master degree at LIRIS laboratory at the University of Claude Bernard, Lyon1 in 2011. Her current research activities are conducted at LIRE laboratory at the University of Abdelhamid Mehri, Constantine 2, Algeria. She participates in national research projects. Her research interests include cloud computing, artificial intelligence, coalitions, green computing and requirements engineering.

1 Introduction

In the midst of the plethora of Cloud Computing definitions, we adopt this definition given by The National Institute of Standards and Technology (NIST): Cloud Computing is a 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 rapidly provisioned and released with minimal management effort or service provider interaction (Mell and Grance, 2011).

From this definition, it is evident that resource allocation poses a significant challenge in Cloud Computing (CC), particularly when dealing with a large number of requests.

Among the issues encountered during the resource allocation, one major concern is energy consumption. In 2014, accounted for nearly 1.62% of the world's total energy consumption, while consumption in 2020 is around 140 billion kWh (Pierson and Hlavacs, 2015). Furthermore, the energy consumption of these data centres is projected to double every five years, posing substantial cost for businesses and the environment. Notably, these data centres are responsible for emitting 2% of CO_2 into the atmosphere (Cotes-Ruiz et al., 2017). Consequently, the concept of *Green Computing* has emerged, emphasising environmental sustainability in data processing, particularly at the data centre level, through the reduction of CO_2 emissions and the utilisation of renewable energy sources. In response to these concerns, researchers have proposed energy-saving solutions while addressing various constraints, such as maintaining Quality-of-Service (QoS) for users, and minimising costs for providers.

The quality of all products often hinges on the ability to meet customer needs. Understanding customer requirements is a critical factor in devising an effective solution. However, stakeholder needs may be influenced by various factors that can change over time, leading to evolving requirements. Studies indicate that changing requirements contribute significantly to project overhauls and are leading cause of project failure, presenting a persist challenge in software development. Identifying expected changes and establishing how to manage them is a major research focus in systems engineering.

Cloud environments are inherently dynamic, making requirements management complex. Thus, addressing system adaptability to changes should be integral part of the development process.

In this regard, it becomes imperative to rely on intelligent agents, enabling Cloud providers to autonomously manage themselves to accommodate dynamic changes, whether initiated by the Cloud provider or customers. The primary challenges in this context consist revolve around: 1/ determining the optimal number of agents for the CC resource management to minimise their quantity while preventing management bottlenecks, and 2/ elucidating the role and behaviour of each agent to ensure effective resource allocation and requirements change management of Virtual Machines (VM) allocation/reallocation requests.

In summary, this paper presents a mechanism that enables the Cloud system to adapt to requirements changes of VM allocation/reallocation requests made by consumers and Cloud providers. The proposed mechanism has a positive impact on the CC system, aiming to maximise Cloud profits by increasing the number of accepted requests, reducing energy consumption and satisfying consumers by upholding QoS and Service Level Agreements (SLA).

Our paper is structured as follows. In the subsequent section, we review related work. Following that, we present our resource allocation mechanism. Then, we outline the steps of the requirements change management process, and in Section 5, we delve into the most significant requirements change scenarios. In the subsequent section, we present a case study. Finally, experimental results are presented in Section 7, and the paper concludes in the final section.

2 Related work

Accepting the maximum of requests is the main objective of Cloud providers. However, the maximisation of accepted requests requires the activation of maximum resources (machines), which subsequently increases energy consumption.

Numerous works have delved into *resource allocation.* For example, Ecarot (2016) focused on optimal and suboptimal allocation of resources in Cloud infrastructures, taking into account both consumers' and providers' interests. A mathematical model for this joint optimisation problem serves as a basis for proposing evolutionary algorithms well-suited for multiple objectives, criteria and constraints.

Song (2022) proposed methods to enhance the utilisation of Cloud infrastructure resources, with focus on the role of pricing in achieving these improvements and leveraging heterogeneity. In the quest to improve utilisation, he explored techniques to optimise network usage through traffic engineering.

To effectively manage Cloud resources, virtual machines can be *migrated* from one physical machine to another. Mahendrabhai (2020) surveyed resource migration using virtual machine in Cloud Computing. This is vital for Cloud providers to efficiently manage and allocate resources as consumer requirements dynamically change. Masdari and Khezri (2020) also an extensive survey and taxonomy of the predictive VM migration approaches adapted for Cloud data centres presented. Kim et al. (2021) introduced a Min-Max Exclusive VM Placement (MMEVMP) strategy to minimise both Service Level Agreements Violation (SLAV) and energy consumption. They compared their strategy with three alternatives: 1) *Random choice*: is a method of randomly selecting a host to which a new VM is assigned. 2) *Low CPU method*: selects hosts with the lowest CPU usage. 3) *Low CPU-Disk method*: opts for hosts with the lowest disk usage from a set of hosts selected by the low CPU method. So, this last algorithm (*low CPU-Disk method*) simply selects hosts with the least CPU and disk usage when the algorithm is called, without considering other factors. In contrast, the MMEVMP strategy selects a host less likely to cause disk SLA Violation (SLAV) from a set of hosts with low CPU usage. The authors demonstrated that the CPU SLAV rate and the disk SLAV rate effectively decreased with the application of their proposed strategy (MMEVMP). They also found a reduction in the number of migrations. However, the MMEVMP strategy requires activating more machines to ensure optimal SLA compliance, resulting in longer total host activation times and increased energy consumption.

None of the aforementioned works prioritise *Green Computing*. Ismaeel et al. (2018) proposed solutions for optimising resource allocation by minimising the energy consumed, as surveyed.

Younis (2022) presented novel resource allocation schemes and computation offloading policies aimed at minimising service latency and enhancing users' Quality of Experience (QoE).

Rebai (2017) addressed profit optimisation for Cloud infrastructure providers engaged in a federation. The overarching goal is to provide effective algorithms for identifying optimal distributed resource allocation plans that strike the best balance between user satisfaction, resource utilisation and cost minimisation within a Cloud federation.

Belghith (2017) proposed formal definitions for Cloud resource management using semantics and social techniques, extending configurable process models to enable Cloud process providers to customise resource allocation based on their requirements.

Shen et al. (2019) focused on enhancing the cloudlet service and proposed an adaptive method of mobile cloudlet to save energy consumption and improve cloudlet utilisation efficiency.

Another work presented in Choukairy (2018), where the author outlined the primary energy reduction techniques in data centres: the DVFS technique, the scheduling process and the migration of virtual machines.

1) *DVFS (Dynamic Voltage and Frequency Scaling):* This technique adjusts the frequency and voltage of the processor of each Cloud server to reduce the energy consumption. Specifically, it is beneficial to reduce the processor's frequency for executing tasks that utilise a low proportion of CPU and a high proportion of memory on the server. If the CPU utilisation rate for a task is high, the task's execution time is short. However, reducing the CPU frequency increases the tasks' execution time because it is inversely proportional to the CPU frequency. Consequently, the percentage of processed tasks decreases, leading to degraded Cloud performance. In cases where a task frequently accesses memory, the processor experiences waiting times. Lowering the CPU frequency during CPU wait times can reduce server power consumption.

2) *Scheduling of tasks:* Involves organising user-generated tasks over time, accounting for time constraints, resource usage and availability constraints. Various scheduling algorithms are employed, such as the Green Scheduler, Round Robin and Random.

    a) *Green scheduler (GS)*: Consolidates tasks onto a minimal number of servers, shutting down unused servers. This algorithm assigns tasks to VMs on the first server and only switches to the next server only if the first one is busy. The algorithm dynamically monitors server availability. For example, the first task goes to the first VM on the first server, and the second and third tasks also go to the first server. However, after these three tasks, the first server becomes saturated, so the fourth and fifth tasks are assigned to the second server.

    b) *Round Robin*: Assigns selected tasks sequentially to available VMs. For example, the first task goes to the first VM of server 1, the second task to the first VM of server 2, the third task is assigned to the first VM of server 3, while the fourth task, fifth and sixth tasks go to the second VM of the first, second and third servers, respectively.

    c) *Random*: Allows tasks to be randomly assigned to VMs.

The author compared these three scheduling algorithms and noted that the Green Scheduler algorithm demonstrates the best power consumption for different data centre workloads compared to the Round Robin and the Random algorithms (more details are provided in Section 8).

3) *Migration of VMs:* Involves transferring VMs from one server to another to reduce the number of servers. Two types of migration exist:

    a) *Cold migration (non-live migration):* Migrates a machine by disabling it on the source server before transfer. After the transfer, the VM becomes active again on the destination server.

    b) *Hot migration (live migration):* Migrates a running VM. Users experience no interruption while accessing the service.

These techniques should be used cautiously as they have limitations. Misusing any of these energy reduction techniques can cause negatively affect the service, leading to performance degradation, which can be problematic. For example, with the DVFS technique, changing frequencies is not instantaneous and the transition from one frequency to another takes time, resulting in energy overconsumption. VM migration may cause performance degradation and consumes energy. Scheduling algorithms can also impact Cloud performance, as is the case with the Random algorithm, which leads to more unprocessed tasks.

To address the limitations of individual techniques, Mishra et al. (2018); Tang et al. (2016) and Wu (2014) proposed combinations of these techniques in pair. Some combine the DVFS technique with scheduling, while others integrate migration with scheduling (Farahnakian et al., 2014; Ghribi et al., 2013).

Choukairy (2018) combined all three techniques: the Green Scheduler algorithm, the DVFS technique and the VM migration.

In our previous works (Bouchareb and Zarour 2019, 2022), we considered the results obtained by Choukairy (2018). We proposed a mechanism based on two thresholds, 'Min and Max'. We addressed SLA compliance and energy consumption issues while applying VM migration to achieve a balanced system. This approach prevents *overloaded* machines to minimise SLA violations (Bouchareb and Zarour, 2019), and reduce energy consumption by avoiding *underloaded* machines (Bouchareb and Zarour, 2022). We primarily detailed the VM migration operation. Our experimental results, as reported in Bouchareb and Zarour (2019, 2022), demonstrate that our proposed mechanism significantly reduces energy consumption.

Many recent works have focused on minimising energy consumption in Cloud Computing. However, in this work we are mainly interested in requirements changes related to Virtual Machines (VM) allocation/reallocation requests, with the aim of maximising the number of accepted requests and minimising energy consumption costs, using agents. Among the works that have introduced the *agent paradigm* in the context of Cloud Computing, we have the Liu (2015) who proposed a customised agent-based reliability monitoring framework to enhance the reliability of Cloud Computing. He developed customised agent architecture, including policy analysis and service critical event prediction for the Cloud Computing environment. This framework utilises temporal

logic to analyse configuration conflicts and predict potential service critical events. He also developed knowledge-augmented temporal logic that incorporates semantic extension in a knowledge base to enhance logical expression and supplement reasoning capability.

Medhioub (2015) defined two agent-based architectures for Federation Clouds. The first architecture combines Cloud Computing and Cloud Networking, introducing the concept of a distributed control plane operating across one or more administrative domains. The second architecture, addresses interoperability and brokerage between Cloud-type services.

For agent-based works focusing on resource allocation in Hadded (2018) is notable. The author first proposed a deterministic optimisation model for allocating an adequate number of Autonomic Managers (AMs) to manage process services. AMs are software agents that implement autonomic behaviour. The second challenge of this work is to optimise the allocation of Cloud resources to host and execute these AMs. However, this work does not consider energy consumption or changing requirements.

In our previous work (Bouchareb et al. 2016), we presented an agent-based architecture and resource allocation policies to increase provider's gain within a Cloud Federation, without addressing the aspect of managing requirement changes.

For works addressing *requirements changes* in the context of Cloud Computing (CC), there are (Bibi et al., 2014) and (Addai, 2020). However, these works do not specifically deal with changing requirements in CC but rather focus on requirement change management in the global software environment using CC.

Zardari and Bahsoon (2011) highlighted the need for a new requirements engineering methodology for businesses and users adopting Cloud services. Existing requirements engineering processes for CC generally address a limited number of non-functional requirements, such as security, privacy and availability. Other recent works have also considered requirements engineering for CC, such as Zalazar (2017). This paper aims to provide a comprehensive and systematic literature review of academic researches conducted in requirements engineering for CC. During this study, some CC approaches were found to consider a limited number of characteristics (e.g., security, privacy, performance) and few activities involving diverse stakeholders. Cloud stakeholders often lack guidelines or standards to manage multiple aspects of services in Cloud environments.

Halboob et al. (2014) proposed a mechanism to dynamically manage changes in Cloud Computing SLAs, but they mentioned that the requirements are specified as high-level requirements, meaning there is no need to specify the new requirements in a new SLA. However, identifying new requirements is a critical phase in the Cloud system, as it is the basis for resource allocation. In our mechanism proposed in this paper, all requirements changes are recorded.

Therefore, existing solutions need improvement because Cloud requirements change unpredictably and continuously due to factors such as new consumer needs, organisational contexts and functionalities. This is why, in a previous work, Bouchareb and Zarour (2021) proposed an agent-based mechanism that manages requirements changes specifically in Cloud federations. In that work, we presented two strategies: *Offer Strategy* and *Acceptance Strategy*, which facilitate the formation of federations (How does a Cloud provider decide to form a new federation? When should they leave an old federation? How to choose the best Cloud providers to form federations? How to decide whether to accept or refuse a federation offer? Etc.) Then, we described how to manage requirements changes in both strategies. So, we addressed requirements changes within Cloud federations.

The objective of the current work was mentioned as a perspective in the previous paper. In this work, we are not concerned with requirements changes in federations but rather with requirements changes related to *customers and Cloud provider requests*: deletion, addition and modification of a VM allocation/reallocation requests. These changes are encountered most frequently and directly impact resource states, necessitating resource reallocation to satisfy customers and minimise energy consumption as much as possible.

In this current work, our focus is on managing internal Cloud resources, not external resources (across other Clouds through Cloud federations – no aspect of federation). The agents involved in these changes are different (as discussed in the next section). Additionally, one of our goals in this paper is to minimise energy consumption, which is why we conducted experiments to test this aspect, along with maximising the profit of Cloud providers (which was not tested in the previous paper).

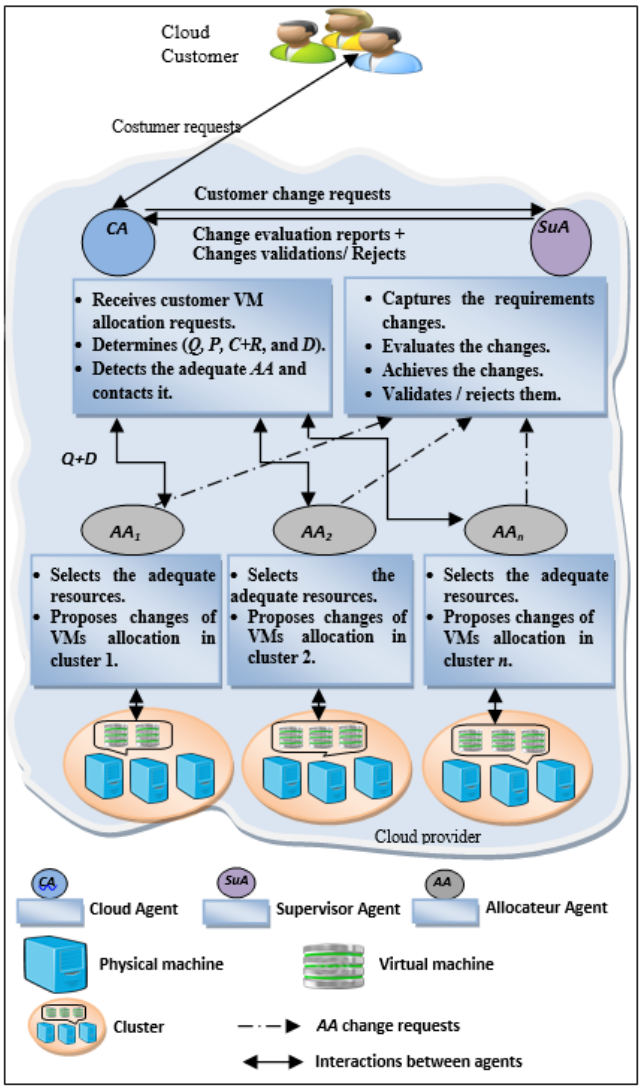
3 The Cloud provider agents

Our architecture is based on three cognitive agents (see Figure 1):

- *Cloud agent (CA)*: Determines the quantity of virtual machines needed for the request (Q), the price (P), the customer's country and region (C+R) and the duration (D). Depending on the request's origin (country and region), *CA* sends the request to the relevant *Allocator Agent* to gain time and reduce energy consumption.
- *Allocator agent (AA):* Upon receiving Q and D, *AA* activates the minimum number of machines required to fulfil the request, aiming to minimise energy consumption and accommodate more requests.
- *Supervisor agent (SuA):* Responsible for capturing requirements changes, identifying the necessary operations to reach the optimal solution.

We are concerned with changes in either '*CA*' (when initiated by the customer) or '*AA*' (when initiated by the Cloud system). Bouchareb and Zarour (2021) focused on requirements changes made by another agent, known as the Coalition Agent '*CoA*' (responsible for forming federations).

Figure 1 The Cloud provider architecture (see online version for colours)



4 The virtual machine allocation mechanism

In this work, the underlying infrastructure is represented by a large-scale Cloud data centre comprising *N* physical machines which can be occupied between 20% and 80%. We have assigned Minimum and Maximum thresholds of '20% and 80%, respectively' to maintain a set of load-balanced machines, especially to minimise energy consumption. The Min and Max threshold strategy has already been studied in Bouchareb and Zarour (2019, 2022). Each machine has a CPU, which can be multicore, with performance measured in Millions of Instructions Per Second (MIPS). Additionally, a machine is characterised by its amount of RAM and network bandwidth. Users submit requests to allocate *M* heterogeneous VMs with resource requirements in terms of MIPS, RAM and network bandwidth. A Service Level Agreement Violation (SLAV) occurs when a VM cannot obtain the requested amount of physical machine resources, often due to VM consolidation. Each VM has a duration that depends on the start and end dates of the executed task.

The Allocator Agent *(AA)* manages a set of nodes within the same region, forming a cluster. When it receives the quantity of VMs necessary to fulfil the request and the usage duration, it selects the appropriate resources based on the following principles:

- A fully charged resource (charge = 80%) consumes less energy than multiple resources with medium or low charge' 'Green Sheduler – explained in section 2' (Younge et al., 2010).
- Aggregate VMs with the same end date onto a single machine to suspend it as soon as possible. This is because the cost of the energy consumption during task execution is directly related to the task's execution time (energy = Power * Time) (Li et al., 2012). Consolidating VMs onto a single physical machine significantly reduces data centre power consumption.

If a migration of some VMs is required, the Supervisor Agent (*SuA)* must:

- Select the appropriate VMs for migration.
- Make decisions about where, when, and how to perform the migration since the migration also consumes energy (Strunk and Dargie, 2013).

5 The steps of the requirements' changes management process

We propose a new mechanism that helps the system automatically make appropriate decisions when it encounters costumers' or internal requirements changes.

The proposed mechanism is inspired by our previous mechanism, which manages requirements changes in CC federations (Bouchareb and Zarour, 2021). This is because:

- The previous proposed mechanism is also an agent-oriented.
- It addresses requirements change management from the early stages of development, recognising that the success of any software solution depends largely on identifying requirements early in the software development (Wiegers and Beatty, 2013).
- It covers requirements change management from the perception of change to its validation.

So, we project this process in our context 'costumers' and internal requirements changes' and not in Cloud federations.

In this work, we have employed the same steps as in the previous mechanism, consisting of five phases to handle any change request (see Figure 2):

1) *Perception*: Identifying the source of the change (internal 'from the Cloud system, by the *CA*', or external 'from the costumer, by the *AA*'), indicating the reason of the change, identifying the stakeholder's requirements that need to be changed (the concerned *AA*), and specifying the type of change (modification / deletion / addition).

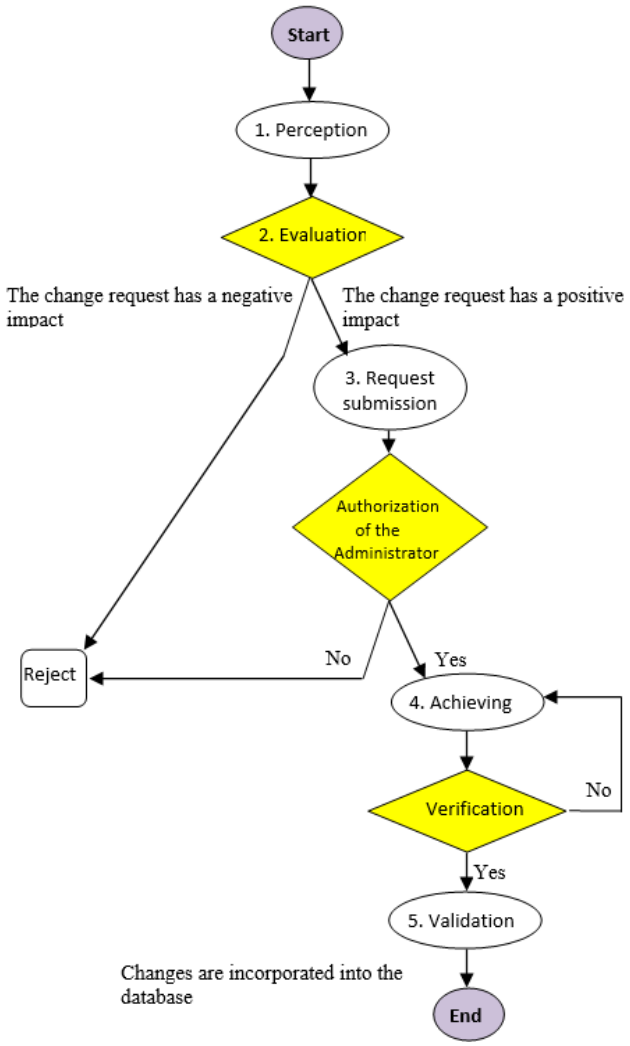
2) *Evaluation*: Decision whether to proceed with the change by evaluating and considering key factors such as time, cost, security, etc.

3) *Request submission*: Providing the evaluation report along with the change request to the *CA*. The *CA* first contacts the administrator to determine whether to approve the change. If approved, the *CA* forwards the request to the *SuA* to achieve it. In some cases, the administrator may authorise direct execution of certain types of changes, such as VM migration or request duration adjustments. However, administrator confirmation is required in other cases, such as cancelling a request and paying penalties to accept a new one).

4) *Achieving change*: Implementing the requested change.

5) *Validation*: Checking (by the *SuA*) that the change request has been executed as specified for validation. If not, 'the achieving change' step may need to be repeated, the requirements of the various system agents are updated accordingly.

Figure 2 The steps of the requirements' changes management process (see online version for colours)



6 Requirements change cases

In this section, we present the most important requirement change cases related to different events: addition, cancellation and modification of requests.

6.1 Adding a request

When a new request is received, it represents a new requirement. For this, it is necessary to include the requirement identifier (*Requi*), request identifier (*Req*), requirement type (*functional*/*non-functional*), the number of the required VMs to fulfil this request, the customer's country and region (*east, west, south, centre, etc.*). Additionally, it is essential to specify the priority level (*high, medium* or *low*), the source (*internal*: if submitted by the Cloud system, *external*: if received from a costumer), request submission date, start and end execution dates, duration and the amount paid by the costumer to fulfil the request.

For this change, an identifier (ID) must be assigned, and the affected requirement identifier, the related request, the type of change (addition, deletion or modification), change duration (temporary/permanent), source, priority, date and description must be provided to explain this change.

Once the request is accepted, it is added to the table containing all accepted requests along with the necessary details.

Accepting a new request will undoubtedly modify at least one resource's state. In the system, there is a table containing all Cloud system resources with their states (free or activated), the number of the occupied VMs on each machine, their occupancy percentage, the executed requests on this resource and the execution duration of each request of each resource.

Adding a new request will increase the load on a machine or even activate a new one. However, as our mechanism is based on 'Green Scheduling' and adapts to changes, some VMs can be migrated from one physical resource to another to minimise energy consumption and, consequently, increase Cloud provider earnings.

6.2 Cancelling a request

In this section, we present cases of request cancellation by the customer or the Cloud Provider (CP):

- *1st case:* The event of customer-initiated request cancellation involves two requirement changes:

  1) *Requirement deletion:* The customer's request is removed from the system.

  2) *Requirement modification:* The states of the physical machines and their VMs will change. The VMs dedicated to this request are released. If the resource is occupied with other requests, it will be partially freed. If the resource is only occupied by this cancelled request, it will be completely freed.

This type of change increases gains because:

1) Liberating VMs reduces power consumption.
2) Cancelling a request by a costumer obliges him to pay penalties.

- *2nd case:* If a new request arrives with significant gains, and the Cloud provider lacks sufficient free resources to accept it, the *SuA* compares the gains of the old requests with this new request. If a less important request is found, its resources are released to satisfy the new one. This obliges the Cloud provider to pay penalties to the customer affected by the cancellation of his request. In this case, we have two permanent changes:
  1) *Requirement deletion:* The old request is removed.
  2) *Requirement addition:* The new request is added to the system.

Adapting to this type of change also increases the provider's gains because it will fulfil the request that offers the maximum gain.

6.3 Modifying a request

We provide examples of cases that trigger modification changes in the Cloud system:

- *1st case:* Similar to the previous case, when the provider releases the resources occupied by a less important request to fulfil the new one. This involves temporary cancellation because the provider postpones the execution of the old request, which will be resumed after the new request is completed. This change is initiated internally by the Cloud system, and the Cloud pays penalties. However, this change maximises the provider's gains because it fulfils both requests.

Another case of modifying requirements but with an *external change source,* where the customer requests to advance or postpone the starting date, ending date, or both. In this scenario, the execution time can change, but the Cloud does not incur any penalties as the change is due to customer request.

If the costumer extends the execution duration of his request, he will pay more. If he shortens it, he will also pay penalties. In both cases, provider gains increase. Additionally, the Cloud gains customer trust and loyalty by successfully accommodating changing requirements.

- *2nd case:* Customers can modify other parameters of their requests, such as changing the storage size or computing power. This alteration affects the required number of VMs for execution, which can increase or decrease. It leads to a permanent modification of the number of VMs and the request cost. Similar to the previous case, we reap the same advantages.
- *3rd case:* If the customer changes his region or country, he sends a request to the Cloud provider to inform it. This modification involves changing the region/country parameter of his requests.

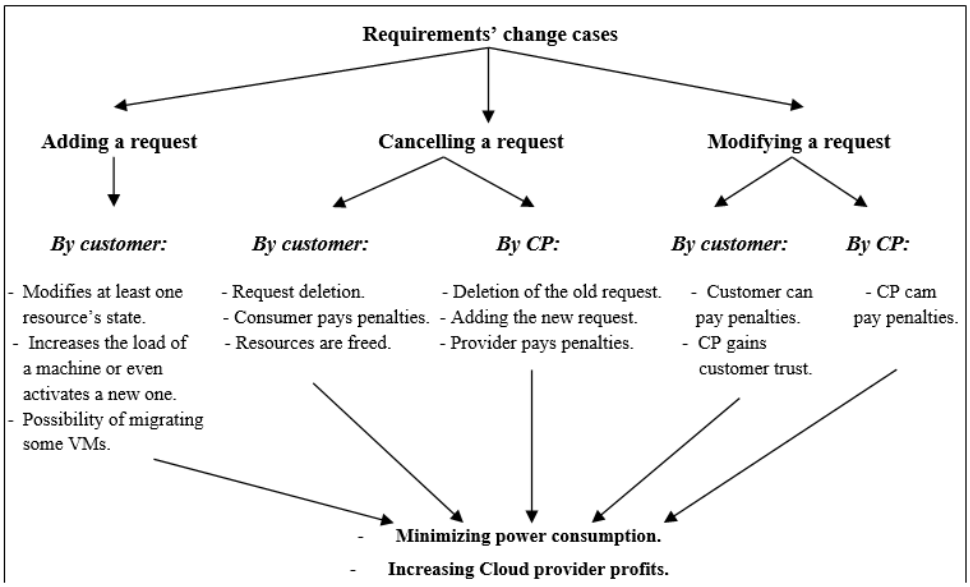
The *SuA* captures the change and migrates the requests to resources closer to the costumer's new location by contacting the appropriate *AA*. This change is essential to minimise response time, communication costs between the costumer and the resources, reduce energy consumption and increase provider gains.

- *4th case:* In this scenario, the destination of VM migration can be modified due to several reasons: the arrival of a new request, the cancellation of an old request, or even a simple modification of another request, such as extending the execution time or increasing the number of necessary VMs.

Here, the change can be permanent or temporary. Sometimes, the request may return to its initial machine, especially when its execution time is long, and the Cloud receives several new requests while completing others. This change aims to minimise the energy consumption and reduce costs.

All requirement change cases are summarised in Figure 3.

Figure 3 Requirements' change cases



7 Use case

7.1 Application context

We are focused on resource allocation and change management to meet customer requirements and maximise Cloud providers' gains. In a Cloud Computing environment, resources are available in various types of VMs.

7.2 Adding a request

Let's assume that our Cloud contains 10 resources; with each resource capable of accommodating a maximum of 8 VMs. Our Cloud has accepted the requests presented in Table 1.

Table 1 Description of all accepted requests

Req	*Source*	*Nbr-VM*	*Country*	*Region*	*Duration*	*Machine (%)*
*Req*1	*C*1	8 VM	Algeria	East	12 M	*R*1 (100%)
*Req*2	*C*2	5 VM	Tunisia	West	6 M	*R*2 (62.5%)
*Req*3	*C*3	6 VM	Morocco	Centre	3 M	*R*3 (75%)
*Req*4	*C*2	2 VM	Tunisia	West	9 M	*R*3 (25%)
*Req*5	*C*2	7 VM	Tunisia	West	5 M	*R*4 (87,5)
*Req*6	*C*1	8 VM	Algeria	East	6 M	*R*5 (100%)

The requests are ordered based on their arrival time in the Cloud system. The Cloud has received a new request, '*Req7*', from costumer *C4,* which requires 8 VMs for a 3-month period with a gain of USD 3000. *C4* is located in Morocco – Centre (see Table 2).

Table 2 Description of request *Req7*

Requi: E7	*Req: Req7*	*Requi type: Functional*	*Nbr-VM: 8 VM*	*Country: Morocco*	*Region: Centre*
Priority: High					
Source: *C*4					
Submission date: 06.30.2022.					
Starting date: 08.01.2022.					
Ending date: 10.31.2023.					
Duration: 3 Months					
Gain: USD 3000					

The satisfaction of this new request, '*Req7*', requires the activation of a new resource, '*R6*'. The latter will be 100% occupied (see Tables: 3, 4, 5 and 6).

7.3 Cancelling a request

We provide examples of the most important cases that trigger a request cancellation change.

- *1st case:* As shown in Table 5, customer *C1* has sent a request, *Req6,* which requires 8 VMs for a duration of 6 months. Then, assuming that *C1* has cancelled this request (see Table 7). The Cloud agent '*CA'* sends this cancellation request to the supervisor agent '*SuA*', which captures the change and triggers it by asking the allocator agent '*AA*' to release the VMs occupied by *Req6*.

Table 3 Description of resources before receiving *Req7*

Resource	*State*	*Nbr VM*	*Percentage*	*Request*	*Duration*
*R*1	Active	8 VM	100%	*Req*1	12 M
*R*2	Active	5 VM	62,5%	*Req*2	6 M
*R*3	Active	6 VM +2 VM	75% | +25%	*Req*3 + *Req*4	3 M + 9 M
*R*4	Active	7 VM	87,5%	*Req*5	5 M
*R*5	Active	8 VM	100%	*Req*6	6 M
*R*6	Free	0 VM	0%	/	/
*R*7	Free	0 VM	0%	/	/
*R*8	Free	0 VM	0%	/	/
*R*9	Free	0 VM	0%	/	/
*R*10	Free	0 VM	0%	/	/

Table 4 Description of change

Change ID: ch07	*Requi: E7*	*Req: Req7*	*Change type: Addition*	*Change duration: Permanent*
Change source: External (C4)		*Description*: Receiving *Req*7 implies **adding** this request to the system and activating resource *R*6.		
Change priority: High	Change date: 08.01.2022			

Table 5 Description of all accepted requests after receiving *Req7*

Req	*Source*	*Nbr-VM*	*Country*	*Region*	*Duration*	*Machine (%)*
*Req*1	*C*1	8 VM	Algeria	East	12 M	*R*1 (100%)
*Req*2	*C*2	5 VM	Tunisia	West	6 M	*R*2 (62,5%)
*Req*3	*C*3	6 VM	Morocco	Centre	3 M	*R*3 (75%)
*Req*4	*C*2	2 VM	Tunisia	West	9 M	*R*3 (25%)
*Req*5	*C*2	7 VM	Tunisia	West	5 M	*R*4 (87,5)
*Req*6	*C*1	8 VM	Algeria	East	6 M	*R*5 (100%)
*Req*7	*C*4	8 VM	Morocco	Centre	3 M	*R*6 (100%)

Table 6 Description of resources after receiving *Req7*

Resource	*State*	*Nbr-VM*	*Percentage*	*Req*	*Duration*
*R*1	Active	8 VM	100%	*Req*1	12 M
*R*2	Active	5 VM	62,5%	*Req*2	6 M
*R*3	Active	6 VM +2 VM	75% +25%	*Req*3 +*Req*4	3 M + 9 M
*R*4	Active	7 VM	87,5%	*Req*5	5 M
*R*5	Active	8 VM	100%	*Req*6	6 M
***R*6**	**Active**	**8 VM**	**100%**	***Req*7**	**3 M**
*R*7	Free	0 VM	0%	/	/
*R*8	Free	0 VM	0%	/	/
*R*9	Free	0 VM	0%	/	/
*R*10	Free	0 VM	0%	/	/

Note: The 100% here represents 80% in reality of the resource load, because in our system if the resource load exceeds 80%, it means that it is overloaded and there is a risk of violation the SLAs. It is then mentioned as 100% to avoid overloading it.

Table 7 Description of change

Change ID: ch01	*Requi: E6*	*Req: Req6*	*Change type: Deletion*	*Change duration: Permanent*
Change source: External (*C*1)		Description: **Cancellation** of the request *Req*6 by costumer *C*1, triggering the deletion of its requirement (*E*6).		
Change priority: High Change date: 08.19.2022				

This change triggers another change at the physical resource, which will be totally freed and then turned off to minimise the energy consumption. Table 8 shows the cancelation of *Req6* from the table containing all the accepted requests.

Table 8 Description of all accepted requests after the change

Req	*Source*	*Nbr-VM*	*Country*	*Region*	*Duration*	*Machine (%)*
*Req*1	*C*1	8 VM	Algeria	East	12 M	*R*1 (100%)
*Req*2	*C*2	5 VM	Tunisia	West	6 M	*R*2 (62,5%)
*Req*3	*C*3	6 VM	Morocco	Centre	3 M	*R*3 (75%)
*Req*4	*C*2	2 VM	Tunisia	West	9 M	*R*3 (25%)
*Req*5	*C*2	7 VM	Tunisia	West	5 M	*R*4 (87,5)
~~***Req6***~~	~~***C1***~~	~~**8VM**~~	~~**Algeria**~~	~~**East**~~	~~**6 M**~~	~~***R5* (100%)~~**
*Req*7	*C*4	8 VM	Morocco	Centre	3 M	*R*6 (100%)

- *2nd case:* We assume that the costumer '*C2'* sends a request, *Req8,* which requires 6 VMs and offers significant gains (USD 8000). However, the Cloud provider does not have enough free resources to accept it. So, the *SuA* compares the gains of the requests that occupy the resources with the new request, '*Req8*'. Assuming that *Req3* offers lower gains (USD 2400). In this case, the *SuA* decides to cancel *Req3* to liberate the resource it occupies (*R3*) and reuse it to satisfy *Req8* (see Tables 9 and 10).

Table 9 Description of change

Change ID: ch02	*Requi : E3*	*Req: Req3*	*Change type: Deletion*	*Change duration: temporary*
Change source: Interne (SuA)		Description: The Cloud provider **temporarily cancels** *Req*3 to satisfy *Req*8, which offers more gain.		
Change priority: High Change date: 07.01.2020				

Table 10 Description of all accepted requests after the change

Req	*Source*	*Nbr-VM*	*Country*	*Region*	*Duration*	*Machine (%)*
*Req*1	*C*1	8 VM	Algeria	East	12 M	*R*1(100%)
*Req*2	*C*2	5 VM	Tunisia	West	6 M	*R*2 (62,5%)
***Req*8**	***C*2**	**6 VM**	**Algeria**	**West**	**6 M**	***R*3 (75%)**
*Req*4	*C*2	2 VM	Tunisia	West	9 M	*R*3 (25%)
*Req*5	*C*2	7 VM	Tunisia	West	5 M	*R*4 (87,5)
*Req*7	*C*4	8 VM	Morocco	Centre	3 M	*R*6 (100%)

This obliges the provider to pay penalties to the customer who submitted the cancelled request, *'C3*' according to the Service Level Agreement. However, with the large gains of *Req8*, the Cloud provider is not a loser even with the payment of penalties.

7.4 Modifying of a request

We give examples of cases that trigger a request modification change.

- *1st case:* We assume that the costumer '*C2*' extends the duration of his request, '*Req2*'. He sends a request to the Cloud provider, identifying the new duration as 9 months instead of 6 months (see Tables 11 and 12).

Table 11 Description of the old request '*Req2*'

Requi: E2	*Req: Req2*	*Requi type: Functional*	*Nbr-VM: 5 VM*	*Country: Tunisia*	*Region: West*
Priority: High *Source*: C2					
Submission date: 05. 05.2022. *Starting date*: 06.10.2022. *Ending date*: 12.09.2022. *Duration*: 6 Months *Gain*: USD 4200					

Table 12 Description of change

Change ID: ch03	*Requi: E2*	*Req: Req2*	*Change type: Modification*	*Change duration: Permanent*
Change source: External (*C*2)		Description: The costumer *C*2 **modifies the duration** of his request from 6 months to 9 months. So, the ending date and the gain of this request will change too.		
Change priority: High Change date: 07.03.2022				

The *SuA* captures the change and executes it (see Table 13).

Table 13 Description of all accepted requests after the change

Req	*Source*	*Nbr-VM*	*Country*	*Region*	*Duration*	*Machine (%)*
*Req*1	*C*1	8 VM	Algeria	East	12 M	*R*1 (100%)
***Req*2**	***C*2**	**5 VM**	**Tunisia**	**West**	**9 M**	***R*2 (62.5%)**
*Req*8	*C*2	6VM	Algeria	West	6 M	*R*3 (75%)
*Req*4	*C*2	2 VM	Tunisia	West	9 M	*R*3 (25%)
*Req*5	*C*2	7 VM	Tunisia	West	5 M	*R*4 (87.5)
*Req*7	*C*4	8 VM	Morocco	Centre	3 M	*R*6 (100%)

- *2nd case:* We assume that *C1* modifies his request, '*Req1*' which requires 8 VMs, by reducing the number of required resources to 6 VMs (see Tables 14 and 15).

Table 14 Description of change

Change ID: ch04	*Requi: E1*	*Req: Req1*	*Change type: Modification*	*Change duration: Permanent*
Change source: External (*C*1)		Description: The costumer *C*1 **modifies the number of required VMs** for his request, *Req*1 from 8VMs to 6VMs. So the gain of this request will change too.		
Change priority: High *Change date*: 06.15.2022				

Table 15 Description of the new request '*Req1'*

Requi: E1	*Req: Req1*	*Requi type: Functional*	*Nbr-VM: 6 VM*	*Country: Algeria*	*Region: East*
Priority: High *Source*: C1					
Submission date: 06. 02.2022. *Starting date*: 06.25.2022. *Ending date*: 06.24.2023. *Duration*: 12 Months *Gain*: USD 10000					

- *3rd case:* We assume that the *C1* changes his location from 'the East' to 'the Centre' of the country. He sends a request to the Cloud provider to modify the execution region of his request, '*Req1*', before its start date (see Table 16).

Table 16 Description of change

Change ID: ch05	*Requi: E1*	*Req: Req1*	*Change type: Modification*	*Change duration: Permanent*
Change source: External (*C*1)		Description: The costumer *C*1 **modifies his location** from East to Centre. So, the region of the allocated VMs of his request, *Req*1, is also modified.		
Change priority: Essential Change date: 08.16.2022				

The *SuA* captures the change, contacts the corresponding *AA* (Allocator Agent responsible for centre resources) to allocate the adequate resources to satisfy the request. If the resources are overloaded, it contacts the neighbouring *AA* at the new location to minimise the response time and the communication costs between the customer and the used resources (see Table 17). The *SuA* agent also contacts the responsible *AA* on the Eastern resources to cancel the request.

Table 17 Description of the new request '*Req1'*

Requi: E1	*Req: Req1*	*Requi type: Functional*	*Nbr-VM: 6 VM*	*Country: Algeria*	*Region: Centre*
Priority: Essential *Source*: *C*1					
Submission date: 06. 12.2022. *Starting date*: 06.25.2022. *Ending date*: 06.24.2023. *Duration*: 12 Months *Gain*: USD 10000					

- *4th case:* We assume that the resource '*R3*' contains the request '*Req4*' which requires only 2 VMs with the end date *ED4* = 03.08.2023. *R3* also contains another request '*Req8*' which will be completed on the date *ED8* = 01.18.2023. There is also the resource '*R2*' which is occupied by the request '*Req2*' which occupies 5 VMs whose end date is *ED2*= *ED4*= 03.08.2023. The *SuA* decides to migrate '*Req4*' to '*R2*' just after the end of '*Req8*' (*ED8*= 01.18.2023) to deactivate resource '*R3*' because it will only be active for the 2 VMs of '*Req4*' (see Table 18).

Table 18 Description of the old migration instruction of *Req4*

Requi: E4	*Req: Req 4*	*Requi Type: Non - Functional*	*Costumer : C2*	*Nbr-VM: 2VM*	*Duration: 3 Weeks*
Instruction: Migration *Initial resource*: *R*3 *destination resource*: *R*2 *Source*: Internal (SuA)			*Description*: Decision to : Migrate*Req*4 from resource *R*3 to resource *R*2 at time 'ED8'. - Switch off *R*3.		
Priority: Essential *Ending date*: 03.08.2023					

The *SuA* decided to migrate '*Req4*' and not '*Req2*' because the system migrates the smallest request, i.e., the one that contains the minimum number of VMs, to facilitate migration and also minimise energy consumption, as migration consumes energy.

Before the arrival of time '*ED8'*, the costumer cancels the request '*Req2*'. This cancellation triggers a change in the migration destination of *Req4* from *R2* to *R5* (see Table 19) to deactivate *R2*.

Table 19 Description of change

Change ID: ch06	*Requi: E4*	*Req: Req4*	*Change type: Modification*	*Change duration: Permanent*
Change source: Internal (SuA)	*Description*: The SuA **modifies the migration destination** of *Req*4 from *R*2 to *R*5.			
Change priority: Essential *Change date*: 01.18.2022				

The *R5* contains 5 VMs with an end date of *ED5* = 03.28.2023. The Cloud system, more precisely the *SuA,* initially chose to migrate to *R2* because its request '*Req2*' ends at the same time as the '*Req4*', this enables the deactivation of resource '*R2*' to save energy and, consequently, increase earnings. With '*R5*', the resource remains activate until *ED5* (03.28.2023) (see Tables 20 and 21).

Table 20 Description of the new migration instruction of *Req4*

Requi: E4	*Req: Req 4*	*Requi Type: Non-Functional*	*Costumer: C2*	*Nbr-VM: 2VM*	*Duration: 3 Weeks*
Instruction: Migration Initial resource: *R*3 Destination resource: *R*5 Source: Internal (SuA)			Description: Decision to: Migrate Req4 from resource *R*3 to resource R5 at time '*ED*8'. Switch off *R*3.		
Priority: Essential Ending date: 03.08.2023					

Table 21 Description of all accepted requests after the change

Req	*Source*	*Nbr-VM*	*Country*	*Region*	*Duration*	*Machine (%)*
*Req*1	*C*1	8 VM	Algeria	East	12 M	*R*1 (100%)
~~*Req*2~~	~~*C*2~~	~~5 VM~~	~~Tunisia~~	~~West~~	~~9 M~~	~~*R*2 (62.5%)~~
*Req*8	*C*2	6 VM	Algeria	West	6 M	*R*3 (75%)
***Req*4**	***C*2**	**2 VM**	**Tunisia**	**West**	**8 M +1 M**	***R*3+*R*5 (25%)**
*Req*5	*C*2	7 VM	Tunisia	West	5 M	*R*4 (87.5%)
*Req*6	*C*4	5VM	Algeria	West	2M	*R*5 (62.5%)
*Req*7	*C*4	8 VM	Morocco	Centre	3 M	*R*6 (100%)

8 Simulation and experimental results

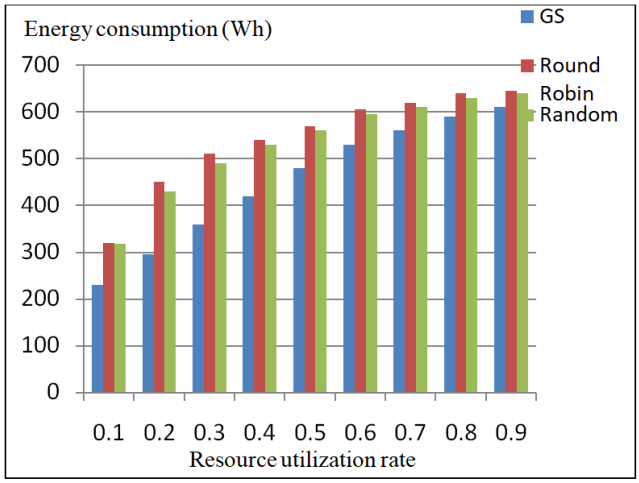
We are now evaluating the performance of the mechanism. We will compare it with some related works to demonstrate the efficiency of our proposed resource allocation mechanism and the necessity of adapting to changing requirements.

The experiment was conducted on a machine with 2.30 GHz Intel® Core ™ i5-6200U CPU, 4096 MB RAM and a 2200 MHz processor speed. To perform our simulation, we use the JADE platform with JAVA as the programming language, NetBeans as the development environment, a relational DBMS 'MySQL' and JDBC interface to establish connection with the database.

In our simulation, physical machines can be occupied between 20% and 80%. Each machine supports 8 VMs; each including compute, storage and network resources. The configuration of the VMs is as follows: 1 CPU core, 8 GB RAM and 500 GB of local storage.

We have compared our mechanism with the work of Choukairy (2018) which we detailed in Section 2. In this work, the author presented the main energy reduction techniques at the data centre. Figure 4 presents the impact of scheduling algorithms: 'Green Scheduler', 'Round Robin' and 'Random' on the energy consumption of servers according to the resource utilisation rate. This rate takes values of: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9 to express the possible load states of a Cloud. We notice that this energy consumption increases as the data centre workload increases. The Green Scheduler algorithm provides the best power consumption for different data centre workloads. This algorithm offers an average reduction of 4% in the energy consumed compared to the Round Robin algorithm and an average reduction of 3.5% compared to the Random algorithm.

Figure 4 Impact of scheduling algorithms on server energy consumption (Choukairy, 2018) (see online version for colours)



To evaluate the overall performance of our algorithm, we first compared the energy consumption of each strategy: Green Sheduler, the proposed mechanism of in Choukairy (2018), which combines the three techniques: the Green Scheduler algorithm, the DVFS technique, and the migration of VMs and our proposed mechanism presented in this paper, which is based on the results obtained by in Choukairy (2018). However, our main focus is on changing requirements.

We compared the energy consumption because it has the greatest effect on overall operating costs. For this, we varied the resource utilisation rate from 0.1 to 0.9. Then, we calculated the energy consumption of servers as well as the percentage of processed tasks in the Cloud, exactly as was done by Choukairy (2018).

Figure 5 presents a comparison between the energy consumption when applying the Green Sheduler algorithm 'GS', the method presented in Choukairy (2018) 'GS+DVFS+Mig', as well as the energy consumption when using our Proposed Mechanism 'PM'. Figure 5 shows that in all three cases, power consumption increases as the data centre workload increases. We find that the method of Choukairy (2018) allows an average reduction of 5% in the energy consumed compared to the use of the GS algorithm. However, our mechanism proposed in this paper allows an average reduction of 10% in energy consumed compared to the use of the GS algorithm and an average reduction of 5% in energy consumed compared to the use of the Choukairy (2018) method. This reduction can reach an additional 5% in certain circumstances. Our mechanism consumes the minimum energy because the allocation of resources, adaptation to requirements changes and VM migration are mainly based on respecting SLAs and minimising energy consumption.

Figure 5 Improving the energy consumption of servers by the proposed method (see online version for colours)

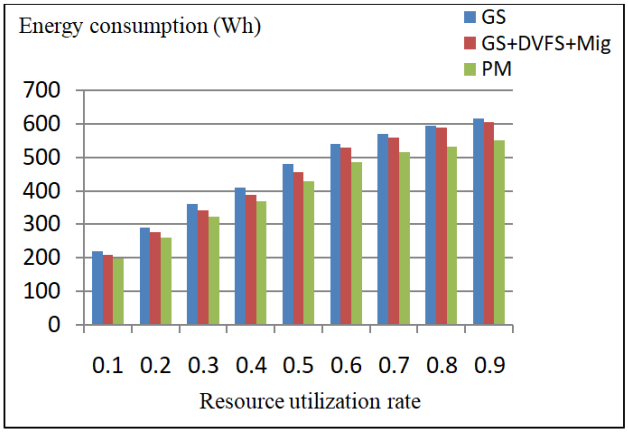
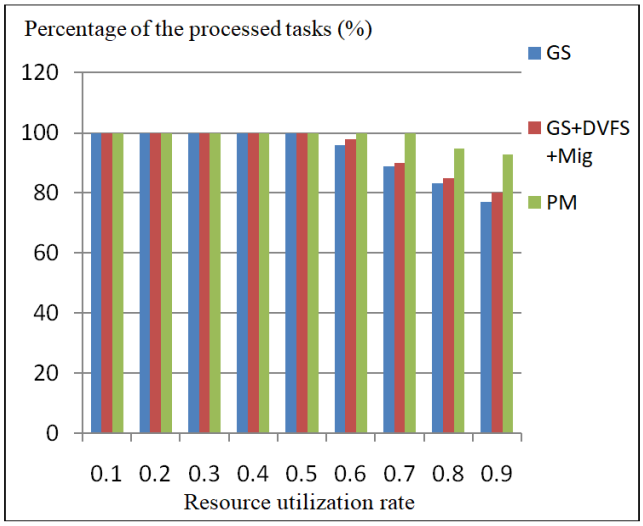


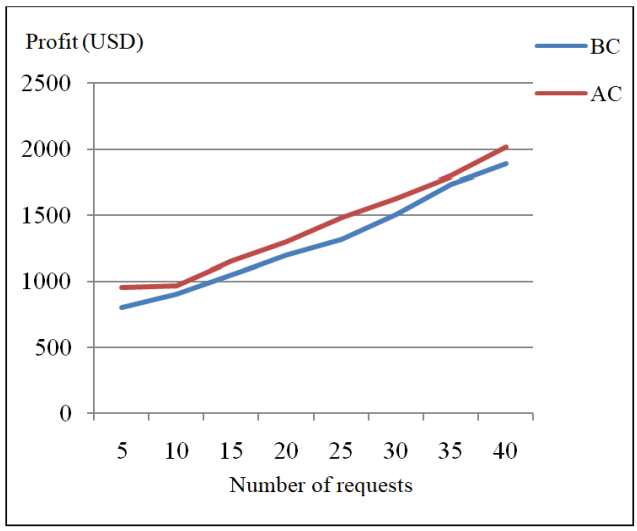
Figure 6 illustrates the percentage of processed tasks using: PM, GS+DVFS+Mig, as well as the percentage of tasks processed by servers when applying the GS. We find that the method of 'GS+DVFS+Mig' provides a slightly higher percentage of processed tasks than that offered by the GS algorithm. However, our mechanism handles the maximum number of tasks. We also notice that the percentages of GS and the method of 'GS+DVFS+Mig' decreases as the resource utilisation rate increases (from 0.6), on the other hand, with our proposed mechanism 'PM', the percentage of processed tasks remains 100% until 0.7, because our mechanism adapts to requirements changes to maximise cloud providers profits while accepting the maximum number of received requests. When the resource utilisation rate is 0.8, the rate drops to only 95% at 0.8 and 93% at 0.9 because the Cloud reaches a stage where, even when activating all the resources, it cannot satisfy all the received requests.

Figure 6 Percentage of processed tasks in our proposed mechanism 'PM', 'GS+DVFS+Mig' and 'GS' (see online version for colours)



Now, we evaluate the Cloud provider's profit before and after adaptating to changing requirements. From Figure 7, we notice that by applying our mechanism, which allows the system to adapt to changes, the Cloud provider gains more than before applying the adaptation to changing requirements. This is attributed to the ability to accept more requests, respecting QoS and SLAs as much as possible, making loyal customers submit even more requests. When the system accepts changes or cancellations of requests, it also allows earning the penalties paid by the customers (as explained in the different cases of our contribution).

Figure 7 Provider profit before and after applying requirement changes (see online version for colours)



9 Conclusions

This paper proposes an agent-based mechanism for managing requirements in Cloud Computing, specifically focusing on Virtual Machine (VM) allocation/reallocation requests in a Cloud Computing system. Our proposed mechanism enables the Cloud system to adapt to the changes, required by the Cloud provider or customers. The main objective of this adaptation to requirements changes is to maximise the gains of Cloud providers by accepting the maximum number of customer requests, retaining them (while respecting QoS and SLAs), and minimising energy consumption (Green Computing).

This paper presents the principle of the virtual machine allocation mechanism and the roles of different agents. It also outlines the steps of the requirements change management process. Furthermore, it details the requirements change cases caused by the Cloud provider or customers, including addition, cancellation and modification of a request. A case study is presented to illustrate the proposed mechanism. In conclusion, our mechanism handles the maximum number of tasks, improves providers' earnings and minimises energy consumption.

In the future, we will test the scalability of our system, involving more received requests, more Cloud providers and more requirements change requests. A second track is to leverage Artificial Intelligence techniques (Machine and Deep Learning) to optimise our solution. We also plan to extend the results of our mechanism to what is called in the literature 'Cloud federation'. We intend to evaluate SLA violations in our mechanism compared to others. The respect of constraints related to various parameters of QoS and SLAs is an important indicator of the effectiveness of the proposed method. Additionally, security is crucial, especially when VMs are migrated from the source machine to the destination machine.

References

Addai, D.A. (2020) *A Cloud Based Framework for Managing Requirements Change in Global Software Development*, Master's thesis in University of Cincinnati, USA.

Belghith, E.H. (2017) *Supporting Cloud Resource Allocation in Configurable Business Process Models*, PhD thesis in the University of Paris-Saclay, France.

Bibi, S., Hafeez, Y., Hassan, M.S., Gul, Z., Pervez, H., Ahmed, I. and Mazhar, S. (2014) 'Requirement change management in global software environment using cloud computing', *Journal of Software Engineering and Applications*, Vol. 7, pp.694–699. Doi: 10.4236/jsea.2014.78064.

Bouchareb, N. and Zarour, N.E. (2019) 'Virtual machines allocation and migration mechanism in green cloud computing', in Chikhi, S., Amine, A., Chaoui, A. and Saidouni, D. (Eds): *Proceedings of the International Symposium on Modelling and Implementation of Complex Systems*, Springer, Vol. 64, pp.16–33. Doi: 10.1007/978-3-030-05481-6_2.

Bouchareb, N. and Zarour, N.E. (2021) 'An agent-based mechanism to form cloud federations and manage their requirements changes', *International Journal of Grid and Utility Computing*, Vol. 12, No. 3, pp.302–321. Doi: 10.1504/IJGUC.2021.10041458.

Bouchareb, N. and Zarour, N.E. (2022) 'VMs migration mechanism for underloaded machines in green cloud computing. In: Bennour, A., Ensari, T., Kessentini, Y. and Eom, S. (Eds): Intelligent Systems and Pattern Recognition', *Proceedings of the International Conference on Intelligent Systems and Pattern Recognition. Communications in Computer and Information Science*, Springer, pp.263–277. Doi: 10.1007/978-3-031-08277-1_22.

Bouchareb, N., Zarour, N. and Aknine, S. (2016) 'Resource management policies to increase provider's gain in a cloud coalition', *International Journal of Grid and Utility Computing*, Vol. 7, No. 3, pp.163–176. Doi: 10.1145/2701126.2701174.

Choukairy, F.E. (2018) *Optimisation de la Consommation D'énergie dans un Environnement Cloud*, Master's Thesis, LAVAL University, Québec, Canada.

Cotes-Ruiz, I-T., Prado, R-P., García-Galán S., Muñoz Expósito, J-E. and Ruiz-Reyes, N. (2017) 'Dynamic voltage frequency scaling simulator for real workflows energy-aware management in green cloud computing', *PloS One*, Vol. 12. Doi: 10.1371/journal.pone.0169803.

Ecarot, T. (2016) *Efficient Allocation for Distributed and Connected Cloud*, Joint PhD Thesis between Telecom SudParis and Pierre and Marie Curie University, France.

Farahnakian, F., Liljeberg, P. and Plosila, J. (2014) 'Energy-efficient virtual machines consolidation in cloud data centers using reinforcement learning', *Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing* (PDP), pp.500–507. Doi: 10.1109/PDP.2014.109.

Ghribi, C., Hadji, M. and Zeghlache, D. (2013) 'Energy efficient VM scheduling for cloud data centers: exact allocation and migration algorithms', *Proceedings of the 13th International Symposium on Cluster, Cloud and Grid Computing* (CCGrid), pp.671–678.

Hadded, L. (2018) *Optimization of Autonomic Resources for the Management of Service-based Business Processes in the Cloud*, PhD Thesis in the University of Paris-Saclay, prepared at Telecom SudParis and the Faculty of Sciences in Tunis, France and Tunisia.

Halboob, W., Abbas H., Haouam, K. and Yaseen A. (2014) 'Dynamically changing service level agreements (SLAs) management in cloud computing', Huang, D.S., Jo, K.H. and Wang, L. (Eds): *International Conference on Intelligent Computing Methodologies*, Springer, pp.434–443. Doi: 10.1007/978-3-319-09339-0_44.

Ismaeel, S., Karim, R. and Miri,A. (2018) 'Proactive dynamic virtual-machine consolidation for energy conservation in cloud data centres', *Journal of Cloud Computing: Advances, Systems and Applications*, Vol. 7, No. 10. Doi: 10.1186/s13677-018-0111-x.

Kim, M.H., Lee, J.Y., R faza Shah, S.A., Kim, T.H. and Noh, S.Y. (2021) 'Min-max exclusive virtual machine placement in cloud computing for scientific data environment', *Journal of Cloud Computing: Advances, Systems and Applications*, Vol. 10, No. 2. Doi: 10.1186/s13677-020-00221-7.

Li, Y., Wang, Y., Yin, B. and Guan, L. (2012) 'An energy efficient resource management method in virtualized cloud environment', *Proceedings of the 14th IEEE International Conference on Network Operations and Management Symposium (APNOMS)*, pp.1–8. Doi: 10.1109/APNOMS.2012.6356086.

Liu, Y. (2015) *Improving Cloud System Reliability using Autonomous Agent Technology*, PhD Thesis in the University of Bridgeport, Connecticut, USA.

Mahendrabhai, P.H. (2020) 'Improve resource migration using virtual machine in cloud computing: a review', *Multidisciplinary International Research Journal of Gujarat Technological University*, Vol. 2, No. 2, pp.81–88.

Masdari, M. and Khezri, H. (2020) 'Efficient VM migrations using forecasting techniques in cloud computing: a comprehensive review', *Journal of Networks, Software Tools and Applications, Cluster Computing*, Vol. 23, pp.2629–2658. Doi: 10.1007/s10586-019-03032-x.

Medhioub, H. (2015) *Federation Architectures and Mechanisms in Cloud Computing and Cloud Networking Environments*, PhD Thesis, Computer Science, Telecommunications and Electronics Doctoral School of Paris, France.

Mell, P. and Grance, T. (2011) *The NIST definition of Cloud Computing*, Recommendations of the National Institute of Standards and Technology, US Department of Commerce Special Publication, Gaithersburg, pp.800–145.

Mishra, S.K., Parida P.P., Sahoo S., Sahoo, B. and Jena, S.K. (2018) 'Improving energy usage in cloud computing using DVFS', in Saeed, K., Chaki, N., Pati, B., Bakshi, S. and Mohapatra, D. (Eds): *Progress in Advanced Computing and Intelligent Engineering. Advances in Intelligent Systems and Computing*, Springer, Singapore, Vol. 563, pp.623–632. Doi: 10.1007/978-981-10-6872-0_60.

Pierson, J-M. and Hlavacs, H. (2015) 'Introduction to energy efficiency in large-scale distributed systems', *Jean-Marc Pierson (edt) Large-Scale Distributed Systems and Energy Efficiency: A Holistic View*, pp.1–16. Doi: 10.1002/9781118981122.ch1.

Rebai, S. (2017) *Resource Allocation in Cloud Federation*, Joint PhD Thesis between Telecom SudParis and University of Pierre et Marie Curie, France. Available online at: https://aisel.aisnet.org/ecis2012/42

Shen, C., Xue, S. and Fu, S. (2019) 'ECPM: an energy-efficient cloudlet placement method in mobile cloud environment', *EURASIP Journal on Wireless Communications and Networking*, Vol. 141. Doi: 10.1186/s13638-019-1455-8.

Song, J. (2022) *Improving Resource Efficiency in Cloud Computing*, PhD Thesis in ST. Louis University, Washington, USA.

Strunk, A. and Dargie, W. (2013) 'Does live migration of VMs cost energy?', *Proceedings of the 27th International Conference on Advanced Information Networking and Applications (AINA)*. Doi: 10.1109/AINA.2013.137.

Tang, Z., Qi, L., Cheng, Z., Li, K., Khan, S.U. and Li, K. (2016) 'An energy-efficient task scheduling algorithm in dvfs-enabled cloud environment', *Journal of Grid Computing*, Vol. 14, No. 1, pp.55–74. Doi: 10.1007/s10723-015-9334-y.

Wiegers, K. and Beatty, J. (2013) *Software Requirements*, 3rd ed., Best Practices, Microsoft, p.637.

Wu, C.M., Chang, R.S. and Chan, H.Y. (2014) 'A green energy-efficient scheduling algorithm using the dvfs technique for cloud data centers', *Future Generation Computer Systems*, Vol. 37, pp.141–147.

Younge, A.J., Laszewski, G.V., Wang, L., Lopez-Alarcon, S. and Carithers, W. (2010) 'Efficient resource management for cloud computing environments', *Proceedings of the 1st International Conference on Green Computing*, pp.357–364. Doi: 10.1109/GREENCOMP.2010.5598294.

Younis, A.N.S. (2022) *Ressource Allocation Ans Computation Offloading in Cloud-Assisted Wireless Networks*, PhD Thesis in the State University of New Jersey, New Brunswick, Canada.

Zalazar, A.S., Ballejos, L. and Rodriguez, S. (2017) 'Analyzing requirements engineering for cloud computing', in Ramachandran M. and Mahmood, Z. (Eds): *Requirements Engineering for Service and Cloud Computing*, Springer Cham, pp.45–64. Doi: 10.1007/978-3-319-51310-2.

Zardari, S. and Bahsoon, R. (2011) 'Cloud adoption: a goaloriented requirements engineering approach', *Proceedings of the 2nd International Workshop on Software Engineering for Cloud Computing*, pp.29–35. Doi: 10.1145/1985500.1985506.