Optimized Service Level Agreement Establishment in Cloud Computing

Leonildo J. M. de Azevedo, Júlio C. Estrella, Luis H. V. Nakamura, Marcos J. Santana, Regina H. C. Santana, Cláudio F. Motta Toledo[1], Bruno G. Batista[2] and Stephan Reiff-Marganiec[3]

[1]*Institute of Mathematics and Computer Sciences, University of São Paulo, SP, Brazil*
[2]*Federal University of Itajuba, Itajuba, MG, Brazil*
[3]*University of Leicester, Leicester, UK*
Email: leonildo.azevedo@usp.br; {jcezar, nakamura, mjs, rcs, claudio }@icmc.usp.br; brunoguazzelli@unifei.edu.br; srm13@leicester.ac.uk

Nowadays, the access to a cloud computing environment is provided on-demand, offering transparent services to clients. Although the cloud allows an abstraction of the behavior of the infrastructure in the service providers (involving logical and physical resources), the Service Level Agreements (SLAs) fulfilment remains a challenge, because depending on the service demand and the system configuration, the providers may not be able to meet the clients requirements. In this way, mechanisms that take account of load balancing and resource provisioning algorithms to provide an efficient load distribution in the available resources are necessary. However, the studies in the literature do not effectively address the problem of the resource provisioning to meet clients requirements using optimization techniques, restricting the analysis to a limited set of objectives. This paper proposes algorithms to address the computational resource provisioning problem using optimization techniques on-the-fly. The techniques optimize the use of the resources available in the cloud infrastructure, aiming to fulfill the clients requirements defined in the SLAs, and ensuring the efficient use of resources.

Keywords: Cloud Computing; Service Level Agreement; Optimization

Received 00 January 2017; revised 00 Month 2017

1. INTRODUCTION

In recent years, cloud computing has been one of the most widely discussed topics in Information Technology (IT). According to the National Institute of Standards and Technology (NIST), the term "Cloud computing" can be defined as follows:

"*Cloud computing is a model for enabling 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*" [1].

Cloud computing can also be regarded as an extension of other paradigms such as standard grade and utilitarian computing; this enables business applications to be viewed as sophisticated services which can be accessed by means of a network (the Internet) [2].

Moreover it involves two essential factors: **Computing** and **Business**. Computing is provided by current technologies such as virtualization, which allows the software and hardware scalability and their use on-demand. However, these resources must be provided in a suitable way and in accordance with the business models established between clients and cloud providers.

Cloud Computing is conceptually divided between three basic services models: **Software as a Service (SaaS)**, **Platform as a Service (PaaS)**, and **Infrastructure as a Service (IaaS)**. These models set out the capacities of the cloud services and how they can be rendered to clients. This approach can be seen as a division of rendered services in abstract layers [3].

The SaaS is concerned with the capacity of the provider to offer applications that can be hosted in a cloud infrastructure [1]. The applications are supplied as a service and they are accessible through a range of client devices. Some examples of these applications, at a business level, include: *Salesforce*, *Google Apps* for personal applications, *Gmail*, *Facebook* and *Twitter* [4].

The use of PaaS is related to the capacity of the provider to enable the client to develop applications in a cloud infrastructure, which are created using

different programming languages, libraries, services and tools supported by the provider. PaaS offers to the developers a platform that facilitates the implementation and installation of applications (Web Applications or SaaS) and, thus avoids the cost and complexity of both purchasing and managing the hardware and software layers [5]. Examples of PaaS are Google App Engine (GAE) and Windows Azure of Microsoft.

IaaS supplies the client with processing capacity, storage, networks and other basic computational resources. The client is able to install and execute software which may include operational and applications systems. The client does not manage or control the infrastructure, but he has control over operational systems, storage, installed applications and a limited control over selected components of the network such as firewalls. The IaaS providers usually offer a virtual infrastructure rather than real hardware. These services can be supplied by standardized interfaces for PaaS and SaaS. Some examples are Amazon Web Services (AWS) with processing services (e.g. Elastic Compute Cloud - EC2) and storage facilities (e.g. Simple Storage Service - S3) [6].

Cloud Computing is a vast and complex computing paradigm that is closely bound to the business dealings of its clients. It has always been a challenging task to provide clients with a suitable infrastructure for rendering services. Different clients require different resources depending on their hosted applications or demands made by users for these applications. For example, a particular client may have a CPU-Bound application while another client has a Memory-Bound application. One client might have an application with a large number of simultaneous accesses, whereas another client might have an application with only a few random accesses.

The Cloud providers generally make several configurations of virtualized resources such as CPU, Memory and Disc which compose the Virtual Machines (VMs) available in the cloud environment. Some configurations of virtual machines are predefined. For example, Amazon works with different classes of VMs[4] and the user can decide what is the best amount and capacity of the virtual machines to ensure that the system operates in a suitable way.

One advantage of Cloud Computing is that the providers offer monitoring services which allow the computational capacity to be extended or reduced. This is carried out on the basis of a parameter configuration and simple scaling policies. Thus, when there is a great demand for services, computational resources can be added automatically to meet the increase in the service demand. On the other hand, if there is a low demand, the resources can be automatically reallocated saving costs for clients, who usually pay per hour. However, these dynamic scaling systems, supplied by providers, take into account only the demand or the rate of use when making a decision about to increase or decrease resources. The exact threshold of when a reshaping of the infrastructure should be carried out is a matter of great interest in academic studies and industry applications.

Another problem is how to quantify the volume of resources (i.e. the number of virtual machines) that must be either initiated or disconnected. It is not a trivial task to carry out this kind of reconfiguration in a run-time, not only with the aim of meeting demands, but also to undertake this procedure in an efficient way. In this context, other objectives must be satisfied such as the reduction of costs, whilst ensuring the quality of services defined by the Service Level Agreement (SLA) for Cloud, as established between provider and clients.

In this paper, our concern is with the IaaS service model and thus we have set out to create and evaluate optimization algorithms that fully comply with SLAs. Three optimization algorithms are proposed aiming to overcome the challenges previously highlighted, based on the problem described in Section 3. First, we describe a deterministic algorithm that is able to search the problem solution space exhaustively. A Simulated Annealing, a Tabu Search and a Multi-Population Genetic Algorithm are also proposed to search the solution space more efficiently. The specific novel contributions of the paper are: 1) an SLA generator that produces sample SLAs based on evaluating proposed property values through simulating respective configurations in CloudSim simulator; 2) a novel Multi-Population Genetic Algorithm (MPGA) that finds optimal SLAs through meta heuristics; and 3) an experimental evaluation of four alternative optimization algorithms.

The paper is structured as follow. A literature review is conducted in Section 2, which addresses optimization within cloud computing. In Section 3, the problem that will be tackled and solved in this study is defined precisely. Section 4, presents another contribution of this study, an SLA generator. Section 5 describes the methods employed for the solution of the problem. In Section 6, the design of the experiments and an analysis of the results achieved by the proposed algorithms are reported. Finally, the conclusions and some guidelines for future work are presented in Section 7.

2. LITERATURE REVIEW

The cloud is a highly scalable environment in which the demand for services can change unexpectedly. Thus, the automatic allocation of resources to meet this demand is becoming an issue of great interest in both the academic and industrial world.

Correct provisioning and resource management allows an efficient use of available computational

[4]https://aws.amazon.com/ec2/instance-types

resources and the whole infrastructure which comprises the Cloud, since it carries out a more efficient mapping between the workload and the resources [7]. For example, the task of providing more computational resources to a client becomes more feasible and they can be provided easily since the resources are virtualized. Moreover, from the standpoint of the clients, the resources can be regarded as unlimited and they can be provided according to the client necessity. However, it should be remembered that the allocation of more resources has an influence on the final cost. This cost has to be passed on to the client and it requires effective management mechanisms on the provider side [8].

There are several studies in the literature that analyze and propose mechanisms for resource management in a cloud environment. However, most of them address the monitoring phase of the SLA. It is important to note that, in this paper, we address the establishment phase and, to date, we have not found papers in the literature that address such problem.

Dynamic policies are extremely important for our study since providers such as Amazon allow the client to define the type of resource scalability that they want to contract by adjusting the number of instances (VMs) that must be allocated or dislocated [9]. In this way, the client is responsible for predicting and defining the best configuration for an efficient resource provisioning in the contracted infrastructure. However, the client prediction and definition of the number of resources cannot be an efficient configuration, harming the QoS (Quality of Service) and the use of the infrastructure.

The studies in the literature that address resource provisioning can be divided into four approaches: based on the policy, based on the heuristic, multiple-criteria and optimization.

The approach based on policy is the simplest approach for resource management, in which the decision is made based on a condition. This approach is used when there is a limited number of scenarios to modify. Thus, this approach is not feasible for the cloud context, due to the high complexity and the various parameters involved in a cloud environment [10] [11].

In the heuristic-based approach, a set of heuristics are previously set to be applied in some scenarios. These strategies are relatively simple, in which various heuristics can be developed and added at run time. However, this approach is limited to the prediction of specific scenarios in a period of time. In this way, SLA violations can occur in scenarios that are not included in the prediction. We find several papers in the literature that apply the heuristic-based approach to the resource provisioning problem. Some studies apply heuristics for automatic initialization of VMs in case of the number of allocated VMs achieves a percentage of utilization [12]. Another study conducted by [13] uses meta-heuristics to reduce the use of resources, aiming at energy saving.

The study carried out by [7] is the study that is closest to our research presented in this paper. The authors proposed a reconfiguration module called ReMM (Resource Management Module). The ReMM seeks to satisfy both clients and providers, maintaining the QoS defined in the SLA and ensuring the efficient use of resources. For this, a heuristic based on current business models is applied changing the number of available resources on-the-fly. However, the authors do not apply optimization techniques.

The optimization approach uses an approach similar to heuristic methods, which is applied in detection and treatment of SLA violations. In this context, the detection can occur through analysis of a system performance model or occurrence of faults, in order to adjust the capacity of the contracted VMs. Usually, optimization approaches use machine learning methods, analysis of temporal series and fault tolerance techniques to detect SLA violations [14] [15] [16]. However, the great complexity of the methods to give an answer is a big problem, resulting in delays to solve specific problems. Therefore, the application of this approach during execution time becomes a challenge.

Finally, multi-criteria, as the name suggests, is an approach that allows the analysis of multiple criteria or situations in a problem [17]. Solutions for the resource provisioning problem based on multi-criteria tend to be decentralized, analysing each early criteria or situation independently [18] [19]. This approach suffers from the same problem of optimization due to its complexity.

There are many complex challenges in cloud computing that can be addressed by optimization techniques. For example, a) the problems in the allocation of virtual machines in a real machine [20]; b) energy saving [21]; c) scalability and load balancing of applications in virtual machines [22]; d) the use of resources to reduce costs, while still guaranteeing a satisfactory performance, the compliance of the SLA and the efficient use of resources [23] [7].

All these problems require optimal or sub-optimal solutions which can be achieved through techniques that are conceptualized in the optimization area. This study seeks to tackle the problem of resources provisioning in a suitable way by employing a heuristic and a meta-heuristic. The proposed methods aim to achieve results that can guarantee the SLA, the system self-management and the QoS for clients of a provider in the cloud.

3. PROBLEM STATEMENT

In a cloud environment, when a well-designed mechanism for provisioning resources is employed, the applications can operate more efficiently with a reduction in costs, a better use of the available infrastructure and a better performance at peak moments when there are variations in the demand for services [7]. However, the provisioning process is complex [24]. According to [25], it requires the definition of better configurations of software and

hardware to ensure compliance with the SLAs as well as to address the need to maximize efficiency and the use of the system.

The management of an SLA is a task composed of several phases, such as negotiation, implementation, monitoring, violation management, reporting, and finalization [26]:

- **Negotiation:** define the terms of services abd include monetary aspects;
- **Establishment:** requests from clients are assigned to the provider resources;
- **Monitoring:** it is important to periodically monitor the resources and the status of the execution;
- **Violation management:** if have a violation, a decision should be take. The decision should be suitable for the context of the violation alert;
- **Reporting and Termination:** provide SLA reports with integrity and provides a method for parties to the agreement terminate the SLA.

Figure 1 illustrates the sequence in which these steps are performed [26]. In this paper, we treat the establishment (implementation) phase, in which the clients requests are set up to the contracted resources (established in the SLA). The ideal scenario is that in which the contracted resources can meet the requirements assigned by the client.

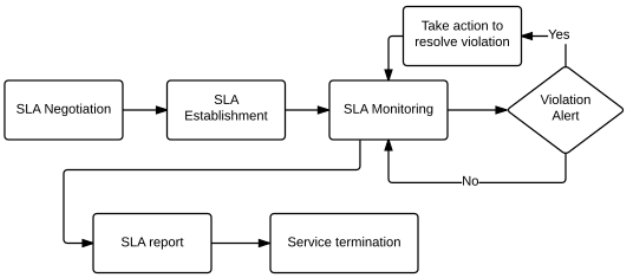

FIGURE 1. SLA management life cylcle [26].

There are several challenges related to provisioning of resources and ensuring requirements using a cloud infrastructure. These problems involve load and performance modelling, virtualization, refinement and monitoring of applications in the virtualized resources [27]. In addition, there are unpredictable circumstances that can impair the efficiency and interfere in the ability to ensure the requirements during the execution time. Among these the following can be cited [28]:

- **Estimation error:** the combination of errors between the computational resources and applications can lead to an under- or over-estimate of client demands, which can have a considerable impact on the contracted QoS and the cost of the service;
- **Dynamic loading:** the Cloud is an environment with different kinds of clients who require different types of services. For this reason, different load peaks can occur depending on the day and time of year or the popularity of an application. These factors give rise to serious problems when estimates are made for the behaviour of the loading and resource definition;
- **Unexpected behaviour:** availability, load, throughput, resource utilization, and network connections can vary in an unpredictable manner in large-scale computing environments such as cloud. This instability in the system can make it difficult to determine the nature of the resources during the provisioning in an efficient way.

Providers such as Amazon EC2 and Microsoft Azure employ a methodology for provisioning resources in which the clients are responsible for giving a precise estimate of the necessary resources and selecting the request to be contracted themselves [29].

It should be remembered that the clients do not always have the technical knowledge to handle the provisioning of resources, whereas this task can be burdensome for them. Therefore, the application of some kind of optimization techniques for the provisioning of resources can make this process an automated task. This can ensure the maximum use of computational resources and hence, the clients do not need to pay for more resources than they really need. They will pay a fair price for the contracted service, with the required QoS.

4. SLA GENERATOR

Considering the papers available in the literature dealing with SLAs, all of them use private instances, i.e., they generate SLAs in accordance with the QoS attributes taken into consideration. Furthermore, these studies use a small amount of SLAs to perform the experiments. For this reason, another contribution of this paper is the development of a SLAs generator.

In this context, based on some related works [7] [26] we identified and used four QoS attributes most common, as follows (note that we use four common factors here, future work will investigate methods to deal with a larger set of factors):

- **Capacity (C):** refers to the number of virtual machines contracted by the client. Three types of VMs are considered: Small, Medium and Large. These instances were modelled based on the configuration of the M3 instances offered by Amazon EC2 (*m3.medium, m3.large and m3.xlarge*)[5]. A data center infrastructure can have a large number of VMs with a wide range of different configurations. The capacity was obtained by Equation 1, in which n is the number of virtual machines:

[5]https://aws.amazon.com/ec2/instance-types/

$$C = \sum_{i=1}^{i=n} Capacity(VM_i) \quad (1)$$

- **Response Time (RT):** refers to the response time of the application expected by the client. This is defined through the execution of the application within the contracted infrastructure. The response time can be obtained by Equation 2 [30]:

$$RT = \frac{\sum_{i=1}^{i=n} ResponseTime(VM_i)}{n} \quad (2)$$

- **Availability (A):** involves an infrastructure that is ready to use. In this context, availability can be obtained in two ways. In the first (Method I), the availability can be calculated as a rate defined by the number of machines considering the probability of some error $(1/n)$. For example, if four VMs are instantiated, the probability of an error occurring would be 1/4 (25%) and the system availability would be 75%. In the second way (Method II), a counting device could be created to find out the percentage of requests met with success and the percentage that failed – the percentage of requests that were successful will be the "**percentage of availability**". In this study, availability will be calculated using both Method I and II. The Method I will be applied in the SLA establishment, while Method II will be applied during the monitoring process, in which the degree of availability can be obtained by Equation 3 [30]:

$$A = \Pi_{i=1}^{i=n} Availability(VM_i) \quad (3)$$

- **Cost per hour (Cost/h):** the monetary value stipulated in the SLA refers to how much the client is going to pay per hour for the service, while making use of the VM. The financial cost per hour can be obtained through Equation 4 [30]:

$$Cost/h = \sum_{i=1}^{i=n} Cost(VM_i) \quad (4)$$

These four QoS attributes allows estimating the minimum and the maximum values for each one. In these way, from these values, a discrete distribution was carried out among them, allowing the combinations between the attributes and the SLAs definition.

The CloudSim simulator was used to model and configure the environment based on the Amazon instances and to generate the SLAs [28]. The client applications simulates a file repository service and a image rendering (more information are presented in Section 5), in which was generated a minimum workload, so that the less powerful instance could process it, and the maximum value was dynamic, varying according to the amount of SLAs generated. Furthermore, an deadline was defined for the application response time, cost per hour, availability and capability.

The intervals follow a discrete order for all attributes and these values were combined to provide the SLAs. The Algorithm 1 describes this implementation and Table 1 shows some examples of SLAs generated.

Algorithm 1 SLA generator

```
1: procedure SLA(Minimum and Maximum values from QoS factors)
2:     for w = minimum cost until w < maximum cost do
3:         for x = minimum workload until x < maximum workload do
4:             for y = minimum response time y < until maximum response time do
5:                 for z = minimum available until z < maximum available do
6:                     accept ← execute(CloudSim[w,x,y,z])
7:                     if accept then SLAs.add(SLA[w, x, y, z])
8:                     end if
9:                 end for
10:            end for
11:        end for
12:    end for//return a SLA list
13: return SLAs
14: end procedure
```

In order to evaluate the SLA generated, in line 5 of the Algorithm 1, CloudSim is executed with the generated QoS factors. If none of the factors exceeds 20% of the value returned by CloudSim, the SLA is added to the list. This test is realised in order to avoid infeasible SLAs [6]. At the work of [7] it is considered a limited of 10%, in this work we decided relax for 20% to test the algorithms comportment and generate more SLAs.

In this paper, we assume that the provisioning of resources should be carried out automatically and dynamically on the basis of the requirements made by clients in the SLA, with a fair price and considering the service demand.

5. METHODOLOGY

In this paper, we proposed an optimization module, which is used in the initial stage of the SLA negotiation, before the provisioning process. Thus, the optimization algorithms are applied to find out the resources that should be allocated to the client, aiming to avoid the SLA violation. Figure 2 illustrates the interaction of the architecture with the optimization algorithm. After the SLA negotiation and resource provisioning, there is the monitoring phase, and in case of violation of the SLA, the optimization framework is triggered and it sets the decision to be taken. This adjustment is not part of the presented work, but is source for future investigation. In this paper, we concentrate on establishing SLAs in the first place.

Problems such as task scheduling and resource provisioning are considered NP-hard [31]. Many problems in this complexity class are solved by integer programming and branch-and-bound approaches [32]. However, these are not suitable to solve decision problems that have continuous adaptation [33].

[6] Are considered infeasible SLAs whether it is impossible to the provider to provide this SLA

TABLE 1. Examples of SLAs generated

	Capacity	Response Time (sec.)	Availability (%)	Cost/h (U$)	Workload**
SLA 1	X*	100	70	0,2	80500
SLA 2	X*	150	70	0,53	100000
SLA 3	X*	180	80	0,6	125000
SLA 4	X*	200	90	0,7	170000

∗The capacity is the goal to be achieved by the algorithms to meet the QoS attributes of the SLA, since the client has insufficient knowledge to precisely make such a decision.
∗∗Cloudlet length: in the CloudSim simulator the workload is given in MIPS (Millions of Instructions Per Second). In this way, the Cloudlet number refers to the number of instructions that the processor is going to execute.

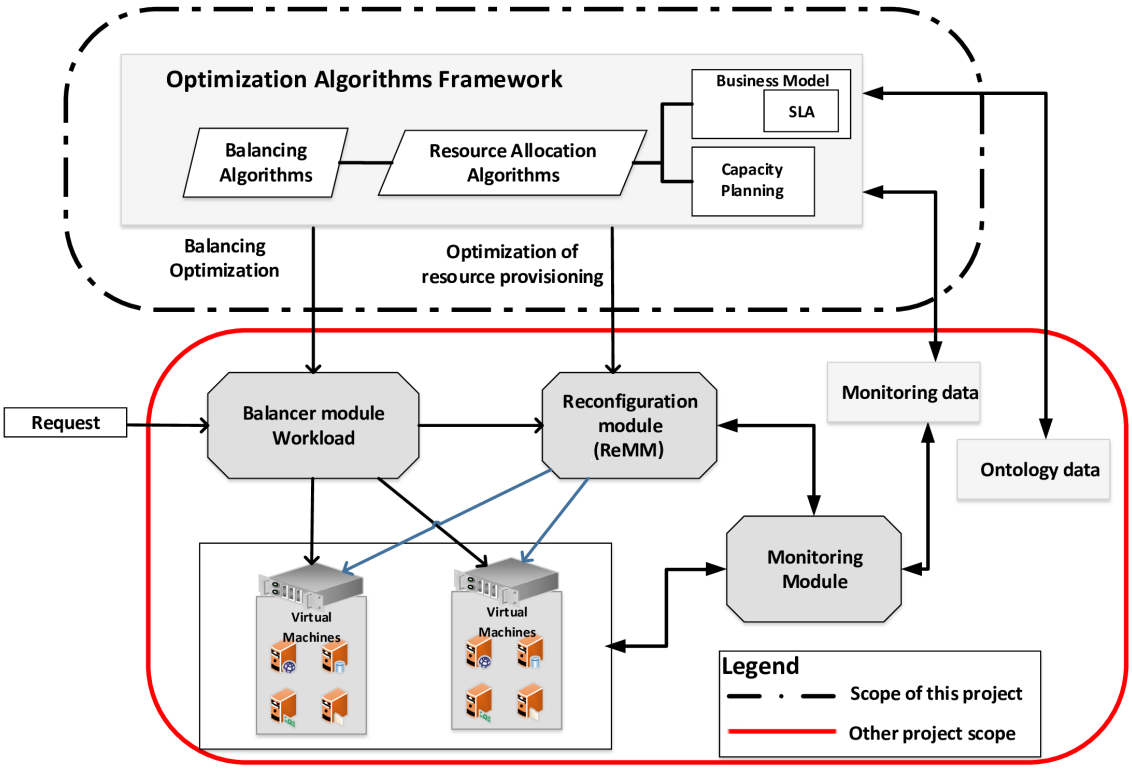

FIGURE 2. Management and provisioning of resources in an architecture with optimization algorithms.

In this context, we have chosen to investigate optimization algorithms based on metaheuristics once these methods present good performance solving real-world problems within a reasonable computation time. There are several metaheurisitcs available in the literature [34], but we decided to apply two local search and one evolutionary algorithm. The option for local search is the need to achieve quickly good solutions. In this case, we will start from one random solution looking for better ones in its neighbourhood. However, we also choose to evaluate the performance of one evolutionary algorithm despite the time spent evolving populations of solutions. Therefore, classical local search techniques such as simulated annealing and tabu search were selected, while the evolutionary algorithm is a multi-population version of the genetic algorithm.

The optimization algorithms were developed with the help of the Professional Optimization Framework (ProOF) tool [35]. The ProOF is intended to guide the implementation of heuristics, metaheuristics, exact and hybrid methods for optimization problems. It provides a framework base for encoding algorithms and setting experiments. The first step was the integration of the ProOF and CloudSim functionalities. Next, the problem was encoded within the framework as well as four optimization methods were designed to solve it: Deterministic Algorithm, Simulated Annealing (SA), Tabu Search (TS) and Multi-Population Genetic Algorithm (MPGA).

These methods aim optimizing the number of virtual machines contracted by the client as described in Section 4. Thus, the representation of the solution (encoding) is defined by a triple (s, m, l), which represents the number of Virtual Machines type Small,

Medium and Large, respectively, that will try to satisfy the clients' requests.

The clients define the desired Capacity (C^c) as well as Time (T^c), Availability (A^c) and Cost/hour (C/h^c). However, it is hard to satisfy all these requirements without conflicts. For example, the requested C^c may not be satisfied with the desired cost C/h^c or may not run an application within time T^c. Therefore, the methods described in this section will try to find the best arrangement of VMs $(s*, m*, l*)$ ables to closely satisfy the values set for C, T^c, A^c and C/h^c.

This is done by evaluating a possible representation (s', m', l') using CloudSim simulator, which will return the related values for $C^{c\prime}, T^{c\prime}, A^{c\prime}, C/h^{c\prime}$. If these parameter values are not compatible with those defined in the SLA (C^c, T^c, A^c, C/h^c), another representation of the solution (s', m', l') must be evaluated. This compatibility is estimated by Equation 5.

$$f(P[i]) = \left|\frac{C^c - C^*}{C^*}\right| + \left|\frac{T^c - T^*}{T^*}\right| + \left|\frac{A^c - A^*}{A^*}\right| + \left|\frac{C/h^c - C/h^*}{C/h^*}\right| \quad (5)$$

The proposed equation is an adaptation of the Manhattan Distance [36] to estimate how close the solution is to the SLA input values. There is a total of four objectives with different scale of values, so Gap values are calculated for each objective.

The first algorithm developed is the so-called deterministic algorithm since it evaluates exhaustively all possible combinations for (s, m, l) as described by Algorithm 2. The deterministic algorithm provides a baseline for performance and results comparison. The method generates and evaluates all possible combinations for triples (s, m, l) from lines 3-7 by keeping track of the best one. The procedure $Generate(s, m, l)$ creates a different combination within the domain of values for (s, m, l) on each iteration, which is evaluated next by $Evaluate(s, m, l)$ using Equation 5. Finally, the best triple (s, m, l) (optimal solution) is returned at line 9.

Algorithm 2 Deterministic algorithm

```
1:  procedure SLA(C^c, T^c, A^c, C/h^c)
2:      //Sweep all the search space
3:      repeat
4:          //generate a capacity to be evaluated
5:          Generate(s, m, l)
6:          Evaluate(s, m, l)
7:      until Until all (s, m, l) possibilities within the
        interval have been generated
8:      //return a better configuration found to satisfy
        the SLA of the client
9:      return SLA* : C*, T*, A*, C/h*
10: end procedure
```

Algorithm 3 describes the MPGA. This method is a genetic algorithm (GA) which operates with a population hierarchically structured in trees as first proposed by [37]. MPGA has been applied to solve different optimization problems in the literature [38, 39, 40, 41] with relevant results reported. Figure 3 illustrates the population structure. The position of the individuals (nodes) in the clusters indicates their position within the hierarchy. In each cluster, the followers has worse fitness value than their leader. Thus, the best individual will be the root in such tree while the worst individuals are the leaves. The method evolves several populations and, after the evolution steps, the best individuals found migrates from one population to the next. Algorithm 3 was based on the hybrid genetic algorithm described in [40].

Algorithm 3 MPGA algorithm

```
1:  procedure MPGA(C^c, T^c, A^c, C/h^c)
2:      for i ← 1 to nPopulation do
3:          InitializePopulation(P)
4:          Evalutate(P[i])
5:      end for
6:      repeat
7:          for i ← 1 to nPopulation do
8:              repeat
9:                  for i ← 1 to P[i].Size*crossRate do
10:                     Selection(P[i])
11:                     Crossover(P[i])
12:                     Mutation(P[i])
13:                     Evaluate(P[i])
14:                     Structure(P[i])
15:                 end for
16:             until P[i] has converged
17:             executeMigration(P[i])
18:             restartPop(P[(i mod nPopulation)+1])
19:         end for
20:     until time limit has been reached
21:     return s
22: end procedure
```

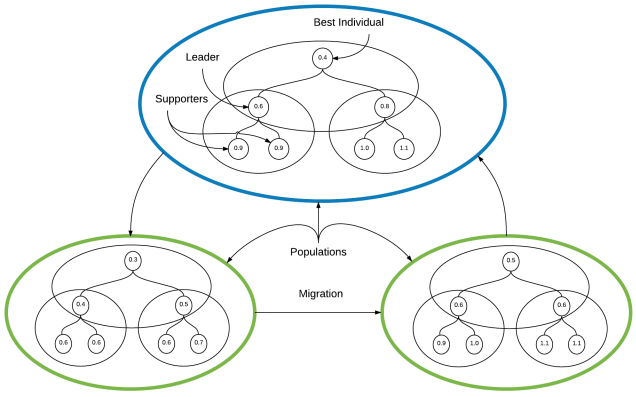

FIGURE 3. Population structure and migration.

Each individual represents a possible VM configuration (s, m, l) for the client. The procedure $InitializePopulation(P)$ generates random individuals

(s, m, l) with $min \leq (s+m+l) \leq max$, where the possible range is defined by $[min, max]$. A total of 5 individuals is generated for each population and evaluated next (lines 2-4). This amount of individuals aims to reduce the fitness evaluation $Evalutate(P[i])$ since it will execute simulations using CloudSim. Next, the evolutionary process starts until convergence has been reached (lines 6-20), generating a total of $P[i].Size * crossRate$ new individuals at each evolution. $Selection(P[i])$ randomly selects a follower individual as one parent and its leader as the other parent. The crossover operator generates a new individual from these two parents, where two operators were implemented: blx-α (blend alpha crossover) and uniform crossover [42].

The uniform crossover exchanges genes between parents, i.e, individuals A and B are selected and each gene in the offspring has 50% of probability to come from parent A or parent B [42]. In Figure 4, the offspring (s', m', l') presents $s' = s^1$,$m' = m^{'}$ from parent A and $l' = m^2$ from parent B. The blx-α crossover defines each gene i by sampling its new value in the range $\alpha \in [0, 1]$ with offspring (s', m', l') given by $s' = \alpha \cdot s^1 + (1 - \alpha \cdot s^2)$, $m' = \alpha \cdot m^1 + (1 - \alpha \cdot m^2)$, $l' = \alpha \cdot l^1 + (1 - \alpha \cdot l^2)$ as shown by Figure 5. One of these two crossover operators is randomly selected each time the crossover must be applied. The new individual may present $(s + m + l) \leq min$ or $max \leq (s + m + l)$, and an adjustment is made over its last value l.

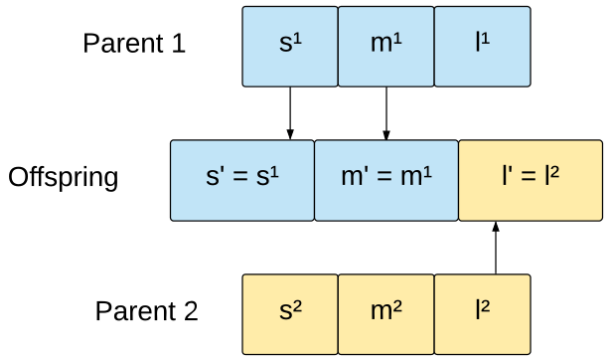

FIGURE 4. Uniform crossover.

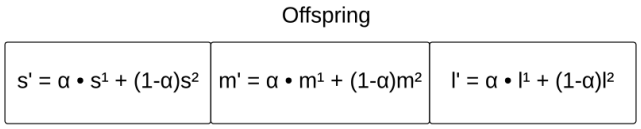

FIGURE 5. Blx-α crossover.

The mutation operator can be applied if mutation rate is satisfied, which means to randomly generate $\lambda \in \{0, 1\}$ with $\lambda \leq mutRate$. In this case, one of the six mutation operators proposed next is randomly selected to be applied:

- **Reset Position:** resets a position of the arrangement (s, m, l);
- **Reset Individual:** similar to reset position, however, in this case resets all arrangements (s, m, l);
- **Swap:** exchanges values of two positions of the arrangement (s, m, l);
- **Proximity:** subtracts the value of a position of the arrangement (s, m, l) and increases in another;
- **Incremental Position:** adds or subtracts the value of a position of the arrangement (s, m, l), respecting the maximum and minimum limit;
- **Incremental Individual:** similar to the incremental position, however, increments or subtracts the value of the all positions of the arrangement (s, m, l).

The new individual is evaluated next and the procedure $Structure(P[i])$ may include it in the hierarchical structure (line 14). A new individual is included when has fitness values better than the worst parent. In this case, $Structure(P[i])$ will also update the new individual position through the tree hierarchy. For instance, if this individual is also better than the best individual found so far, it will be rearranged to become the root node in the tree.

The evolutionary steps carried on population $P[i]$ converge when no new individual is inserted after $P[i].Size * crossRate$ attempts. At this point, a copy of the best individual of $P[i]$ is sent by executeMigration(P[i]) to the next population to be evolved. Finally, $restartPopulation(P[(imodnPopulation) + 1])$ restarts the next population, except by its best individual and the migrated one. MPGA stops when the time limit is reached.

The second algorithm is an adaptation of the Simulated Annealing proposed by [43]. This method was developed based on the thermodynamic principles acting as a local search technique. Solutions in the neighbourhood of the current one can be evaluated through a temperature function $T = \frac{T}{1+\alpha*\sqrt{T}}$, in which α is a variable of the method acting as the cooling reason. There are also other parameters of control, such as the number of iterations $(SAmax)$ to look for solutions and the initial temperature (T_0). Algorithm 4 shows the proposed SA. In this case, s stands for (s, m, l) and $f(P[i])$ (Equation 5) is represented by $f(s)$.

For each temperature T, a total of $SAmax$ solutions is generated in the neighbourhood of the current solution (s). The procedure $generateNeighbour(s)$ randomly select one of the six mutation operators, previously described for the MPGA, to generate a neighbouring solution s' from s. If the solution s' is better (lines 9-11), it becomes the current solution s (line 12). Such solution may also update the best one found so far (lines 13-14). Otherwise, there is a chance to accept a worse neighbouring solution s' as current one, based on Δ and T values by following expression $e^{-\Delta/T}$ (lines 17-20). This likelihood is larger during high temperature T, which decrease using the cooling

Algorithm 4 SA algorithm

```
1: procedure SA(C^c, T^c, A^c, C/h^c)
2:     s* ← s // better solution
3:     T ← T_0 // current temperature
4:     IterT ← 0 // number of iteration in T
5:     repeat
6:         while T > 0.001 do
7:             while IterT < SAmax do
8:                 IterT ← IterT + 1
9:                 s' ← generateNeighbour(s)
10:                Δ ← f(s') − f(s)
11:                if Δ < 0 then
12:                    s ← s'
13:                    if f(s') < f(s*) then
14:                        s* ← s'
15:                    end if
16:                else
17:                    x ∈ [0, 1]
18:
19:                    if x < e^{−Δ/T} then
20:                        s ← s'
21:                    end if
22:                end if
23:            end while
24:            T ← T / (1 + α * √T)
25:            IterT ← 0
26:        end while
27:        s ← s*
28:        T ← T_0
29:    until time limit has been reached
30:    return s
31: end procedure
```

Algorithm 5 Tabu Search algorithm

```
1: procedure Tabu(C^c, T^c, A^c, C/h^c)
2:     s* ← s
3:     repeat
4:         BetterIter ← 0
5:         Iter ← 0
6:         TL ← ∅
7:         while Iter − BetterIte ≤ TSmax do
8:             Iter ← Iter + 1
9:             s' ← selectBestNeighbor(s)
10:            Δ ← f(s') − f(s)
11:            if Δ < 0 then
12:                s ← s'
13:                Update(TL)
14:                if f(s) < f(s*) then
15:                    s* ← s
16:                    BetterIter ← Iter
17:                end if
18:            end if
19:        end while
20:        s ← s*
21:    until time limit has been reached
22:    return s
23: end procedure
```

reason (line 24). If the time limit has not been reached, the temperature is reheated and the search continues in the neighbourhood of the current solution (lines 27-28).

Tabu Search (TS) is another local search method proposed by [44] with a memory structure that allows to escape from local optimum values. This can be done by forbidding search movements recently applied which can lead to neighborhoods previously explored. The mutation operators from MPGA are also applied here as movements of TS to generate neighbouring solutions. Algorithm 5 describes the method. In this case, s represents (s, m, l), $f(P[i])$ (Equation 5) is replaced by $f(s)$, $BetterIter$ is the last iteration where the best solution was updated, TL stands for tabu list and $BTmax$ is the maximum number of iterations without improvement in the best solution.

The local search goes on while the time limit is not reached and there is a recent improvement over the best solution found so far (lines 3-7). Initially, the six mutation operators are applied over the current solution s and the best neighbouring solution s' found is selected next (line 9). If s' is better than s, the current solution is replaced by s' and the movement (mutation operator) that generates s' becomes tabu (line 13). Thus, this movement can not be applied again to generate a neighbouring solution. If the size of TL is exceeded, the FIFO (First In, First out) police is used to insert and remove movements from it. The parameter $BetterIter$ is set by the current iteration $Iter$ when a new best solution is reached, so it becomes possible to continue the local search in the neighbourhood of $s*$ (lines 14-16).

TABLE 2. Specification of instances

Instances	Virtual Core	Main Memory (GB)	Disk SSD
m3.medium	1	3.75	1 x 4
m3.large	2	7.5	1 x 32
m3.xlarge	4	15	2 x 40

6. COMPUTATIONAL RESULTS

The aim of the experiments shown in this section is to analyze which computational resources should be provided to fulfilment the SLA. The experiments were carried out in the CloudSim Simulator 3.0.3 3 version[7], with the aid of a computer with an AMD Phenom(tm) II X6 1090T Processor, 16 GB of RAM memory, 1.5 TB of disc storage and the Ubuntu 14.04.3 LTS operational system with a kernel version 3.13.0.

The environment was configured for the execution of three types of VM instances described in Table 2, modelled based on Amazon instances.

Two benchmarks were modelled in the experiments to simulate a file repository service (Apache Benchmark [45]) and a image rendering (Smallpt Benchmark [46]),

[7] http://www.cloudbus.org/cloudsim/

respectively. Furthermore, we used the SLA generator described in Section 3, which generated 131 SLAs.

The first experiment evaluates the performance of the deterministic method. The method was executed in 10 scenarios for a client SLA, ranging from data centers with 10 to 100 VMs. The aim is to find the best set of resources configuration, satisfying the SLA or at least returning the closest result of the best one if the SLA is infeasible. Owing to this method ensures the best possible solution set, the response variable was the response time. Figure 6 shows the results. This experiment was executed with Apache and Smallpt, however, for both benchmarks the determinist algorithm obtained the same result, i.e, the workload does not influence on the algorithm performance. The optimal solution is always obtained, but the deterministic method takes polynomial time which makes it infeasible for a commercial application.

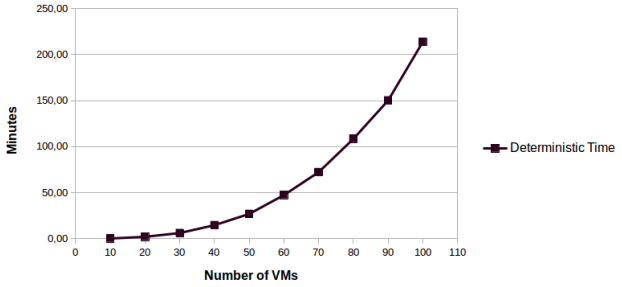

FIGURE 6. Response time of the deterministic model.

To evaluate the other methods, CloudSim was configured to run with a limited amount of resources (100 VMs), divided evenly between VMs of small, medium and large types. In this way, it is possible to simulate a real cloud environment. For experiments in this environment, the SLA generator creates 100 clients that allocate resources in the cloud at different times (5 clients by time), as shown in Figure 7. As clients establish the SLAs, the optimization algorithms are executed to determine the best configurations.

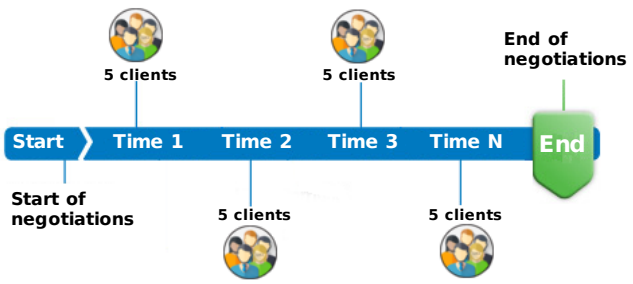

FIGURE 7. Timeline of SLA negotiations between client and provider.

For each set of clients, the optimization algorithms were executed 10 times and the solution with the best fitness was chosen. The execution was carried out in a distributed way (simultaneously) and each algorithm has been implemented and evaluated separately. That is, the same experiment was replicated for all algorithms. Table 3 shows the settings for each algorithm. These values were empirically defined, based on previous settings reported in the literature for such methods ([40]) and best results achieved from some tuning tests executed by us. Figure 8 brings the results for each experiment in the Apache Benchmark.

TABLE 3. Algorithms settings

Characteristics	SA	Tabu	MPGA
Time execution (minutes)	5	5	5
Temperature (max-min)	1000-0.001	-	-
alpha	0.95	-	0.2
Operators	All	All	All
SAmax	3	-	-
TSmax	-	10	-
TL	-	1	-
Neighborhood	-	5	-
Tabu Max	-	1	-
Individuals	-	-	5
Populations	-	-	3
Mutation rate	-	-	0.7
Crossover rate	-	-	5.0

Note that in Figure 8, with the exception of SA method, the lines are overlapped, which means that the tabu search and MPGA found the optimal solution. However, at a closer look obtained by expanding a random section of the graph, it becomes visible that the tabu search does not arrive at optimum, but found an almost as good solution.

Figure 9 shows the results obtained for each experiment in the Smallpt benchmark. In this experimentation it is possible see that the behaviour of determinist and MPGA algorithm does not change, while SA has the worst performance among them.

Another analysis conducted between the methods was to assess the reliability of answers, because all methods are stochastic. For this, we randomly extracted a sample of 5 clients from the 100 evaluated. As each client was run 10 times for each method, we got the full set of data to analyse.

In Figure 10, we show a blox pot graph, where are the results of this analysis. The center of the distribution is indicated by the median line in the center of the square, the dispersion is represented by the amplitude of the graph and outliers appear as distant points. Therefore, it is possible to see that the MPGA obtained the most reliable results, reaching the best results in all executions. The tabu search was almost as good as MPGA, however presented some outliers, while simulate annealing had several variations.

Owing to the MPGA method obtained the best results, we conducted a convergence time analysis, in which we took again five random clients. Figures 11 and 12 show the results of this analysis to the MPGA and Tabu search. The convergence time of the MPGA ranged between 45 and 135 seconds; a shorter time than 5 minutes defined as stop criterion and significantly lower than the 3.5h taken by the deterministic method.

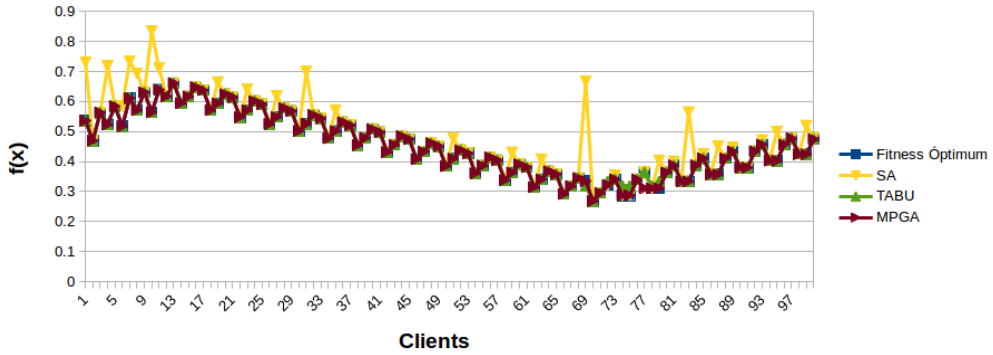

FIGURE 8. Performance of simulate annealing, tabu search and MPGA in relation to the deterministic method with the Apache Benchmark.

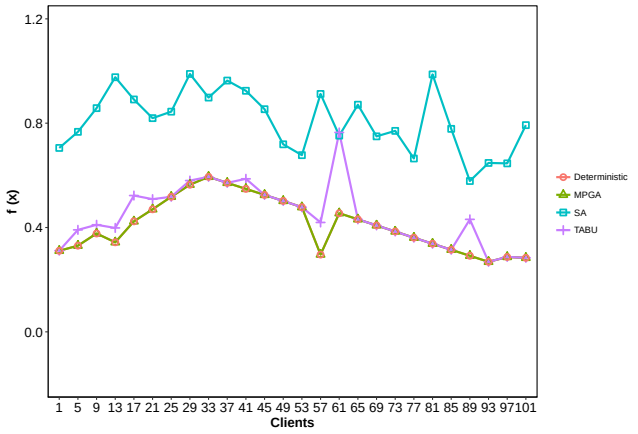

FIGURE 9. Performance of simulate annealing, tabu search and MPGA in relation to the deterministic method with Smallpt benchmark.

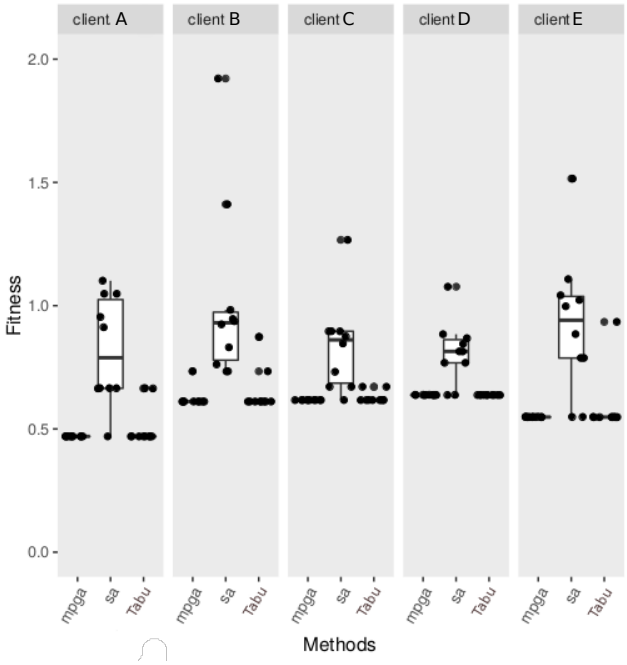

FIGURE 10. Reliability analysis of the methods.

In the other hand, Tabu search has worse convergence results, where it does not found the optimal solution to client 1 and 3.

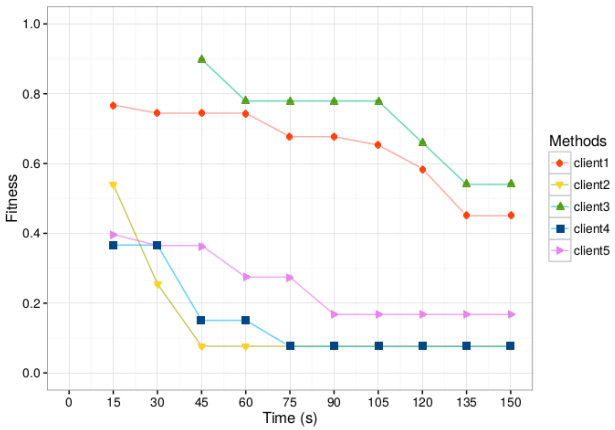

FIGURE 11. Convergence analysis of the MPGA.

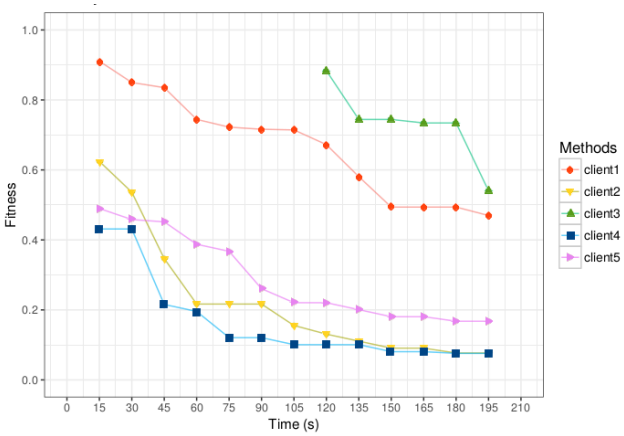

FIGURE 12. Convergence analysis of the Tabu.

7. CONCLUSIONS

Providing service to the clients of the Cloud with an efficient infrastructure, that respects the SLA and

its QoS attributes, while at the same time trying to reduce costs, is not a trivial task. Within the domain of cloud computing, the most wide-ranging problems can be mapped out in solutions that generally involve optimization based on their complexity and the large number of resources that can be scalable.

This paper addressed one of these challenges, namely to provide compliance with the SLAs defined between clients and providers. Our proposal mapped some of the QoS attributes that determine the criteria for the SLA and based on this, we designed and analyzed algorithms that allow an optimized configuration of the cloud infrastructure. The results evidenced that the algorithm based on meta-heuristics Tabu search and MPGA were efficient and applicable as solutions to the problem.

As future work, we intend to implement the second stage of this study, that is the monitoring phase, in which, if the SLA is not being fulfilled, the renegotiation process between the client and provider will be considered to establish a new infrastructure configuration. Other QoS attributes will be added in the SLA and new constraints to the problem, such as defining a maximum cost threshold that the client is able to afford.

ACKNOWLEDGEMENTS

Authors would like to thank the financial support from CNPq, CAPES and FAPESP (processes IDs: 11/12670-5, 15/11623-4 and 16/14219-2) for funding the bulk of this research project.

REFERENCES

[1] Mell, P. and Grance, T. (2017). The nist definition of cloud computing. Available at: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. Last Access: 06/27/2017.

[2] (2009) Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, **25**, 599 – 616.

[3] Buyya, R., Broberg, J., and Goscinski, A. M. (2011) *Cloud Computing Principles and Paradigms*. Wiley Publishing.

[4] Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., and Ghalsasi, A. (2011) Cloud computing - the business perspective. *Decision Support Systems*, **51**, 176 – 189.

[5] Intel-Corporation (2017). Cloud computing taxonomy and ecosystem analysis. Available at: http://www.intel.com/content/dam/doc/case-study/intel-it-cloud-computing-taxonomy-ecosystem-analysis-study.pdf. Last Access: 06/27/2017.

[6] Patidar, S., Rane, D., and Jain, P. (2012) A survey paper on cloud computing. *Advanced Computing Communication Technologies (ACCT), 2012 Second International Conference on*, pp. 394–398.

[7] Batista, B. G., Estrella, J. C., Ferreira, C. H. G., Leite Filho, D. M., Nakamura, L. H. V., Reiff-Marganiec, S., Santana, M. J., and Santana, R. H. C. (2015) Performance evaluation of resource management in cloud computing environments. *PloS one*, **10**, 21.

[8] Coutinho, E. F., de Carvalho Sousa, F. R., Rego, P. A. L., Gomes, D. G., and de Souza, J. N. (2015) Elasticity in cloud computing: a survey. *annals of telecommunications-annales des télécommunications*, **70**, 289–309.

[9] Amazon (2017). Dynamic scaling. Available at: http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-scale-based-on-demand.html. Last Access: 06/27/2017.

[10] Huebscher, M. C. and McCann, J. A. (2008) A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys (CSUR)*, **40**, 7.

[11] Yuchao, Z., Bo, D., and Fuyang, P. (2012) An adaptive qos-aware cloud. *Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on*, pp. 160–163.

[12] Emeakaroha, V. C., Brandic, I., Maurer, M., and Breskovic, I. (2011) Sla-aware application deployment and resource allocation in clouds. *Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual*, pp. 298–303.

[13] Kessaci, Y., Melab, N., and Talbi, E.-G. (2014) A multi-start local search heuristic for an energy efficient vms assignment on top of the opennebula cloud manager. *Future Generation Computer Systems*, **36**, 237–256.

[14] Eyraud-Dubois, L. and Larchevêque, H. (2013) Optimizing resource allocation while handling sla violations in cloud computing platforms. *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pp. 79–87.

[15] Jiang, J., Lu, J., Zhang, G., and Long, G. (2013) Optimal cloud resource auto-scaling for web applications. *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pp. 58–65.

[16] Wang, X., Du, Z., and Chen, Y. (2012) An adaptive model-free resource and power management approach for multi-tier cloud environments. *Journal of Systems and Software*, **85**, 1135–1146.

[17] (2010) Multi-criteria decision making approaches for supplier evaluation and selection: A literature review. *European Journal of Operational Research*, **202**, 16 – 24.

[18] Yazir, Y. O., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., and Coady, Y. (2010) Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pp. 91–98.

[19] Yazir, Y. O., Akbulut, Y., Farahbod, R., Guitouni, A., Neville, S. W., Ganti, S., and Coady, Y. (2012) Autonomous resource consolidation management in clouds using impromptu extensions. *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp. 614–621.

[20] Masdari, M., Nabavi, S. S., and Ahmadi, V. (2016) An overview of virtual machine placement schemes in cloud computing. *Journal of Network and Computer Applications*, **66**, 106–127.

[21] Beloglazov, A., Abawajy, J., and Buyya, R. (2012) Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, **28**, 755 – 768.

[22] L.D., D. B. and Krishna, P. V. (2013) Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*, **13**, 2292 – 2303.

[23] Byun, E.-K., Kee, Y.-S., Kim, J.-S., and Maeng, S. (2011) Cost optimized provisioning of elastic resources for application workflows. *Future Generation Computer Systems*, **27**, 1011 – 1026.

[24] Jennings, B. and Stadler, R. (2015) Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, **23**, 567–619.

[25] Guzek, M., Bouvry, P., and Talbi, E.-G. (2015) A survey of evolutionary computation for resource management of processing in cloud computing. *Computational Intelligence Magazine, IEEE*, **10**, 53–67.

[26] Faniyi, F. and Bahsoon, R. (2016) A systematic review of service level management in the cloud. *ACM Computing Surveys (CSUR)*, **48**, 43.

[27] Mustafa, S., Nazir, B., Hayat, A., Madani, S. A., et al. (2015) Resource management in cloud computing: Taxonomy, prospects, and challenges. *Computers & Electrical Engineering*, **47**, 186–203.

[28] Calheiros, R. N., Ranjan, R., and Buyya, R. (2011) Virtual machine provisioning based on analytical performance and qos in cloud computing environments. *Parallel Processing (ICPP), 2011 International Conference on*, pp. 295–304.

[29] Huu, T. T. and Montagnat, J. (2010) Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pp. 612–617.

[30] Ko, J. M., Kim, C. O., and Kwon, I.-H. (2008) Quality-of-service oriented web service composition algorithm and planning architecture. *Journal of Systems and Software*, **81**, 2079–2090.

[31] Kumbhare, A., Simmhan, Y., and Prasanna, V. K. (2013) Exploiting application dynamism and cloud elasticity for continuous dataflows. *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* 57.

[32] Woeginger, G. J. (2003) Exact algorithms for np-hard problems: A survey. *Combinatorial Optimization—Eureka, You Shrink!*, pp. 185–207.

[33] Kumbhare, A. G., Simmhan, Y., Frincu, M., and Prasanna, V. K. (2015) Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure. *Cloud Computing, IEEE Transactions on*, **3**, 105–118.

[34] Glover, F. and Kochenberger, G. (2006) *Handbook of Metaheuristics* International Series in Operations Research & Management Science.

[35] Arantes, M. d. S. (2014) Ambiente para desenvolvimento de métodos aplicados a problemas de otimização. Master's thesis. Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo.

[36] Black, P. E. (2004) *Dictionary of algorithms and data structures.* National Institute of Standards and Technology Gaithersburg.

[37] Franca, P. M., Mendes, A., and Moscato, P. (2001) A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research, v*, **132**, 377–2217.

[38] L. Buriol, P. M. F. and Moscato, P. (2004) A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics, Kluwer Academic Publishers, Hingham, MA, USA*, **10**, 1381–1231.

[39] Moscato, P., Mendes, A., and Berretta, R. (2007) Benchmarking a memetic algorithm for ordering microarray data. *Biosystems*, **88**, 56 – 75.

[40] Toledo, C. F. M., da Silva Arantes, M., De Oliveira, R. R. R., and Almada-Lobo, B. (2013) Glass container production scheduling through hybrid multi-population based evolutionary algorithm. *Applied Soft Computing*, **13**, 1352–1364.

[41] da Silva Arantes, J., da Silva Arantes, M., Toledo, C. F. M., Júnior, O. T., and Williams, B. C. (2017) Heuristic and genetic algorithm approaches for UAV path planning under critical situation. *International Journal on Artificial Intelligence Tools*, **26**, 1–30.

[42] Eiben, A. and Smith, J. (2007) *Introduction to Evolutionary Computing* Natural Computing Series. Springer Berlin Heidelberg.

[43] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983) Optimization by simmulated annealing. *science*, **220**, 671–680.

[44] Glover, F. (1986) Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, **13**, 533–549.

[45] Apache benchmarking. Available from: https://openbenchmarking.org/test/pts/apache. Accessed: 06-27-2017.

[46] Smallpt benchmarking. Available from: https://openbenchmarking.org/test/pts/smallpt. Accessed: 06-27-2017.

[47] Takahashi, M. and Kita, H. (2001) A crossover operator using independent component analysis for real-coded genetic algorithms. *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, pp. 643–649. IEEE.